Parallélisation de simulateur DEVS par métamodélisation et transformation de modèle

Hamidou Togo

N° d'ordre : D. U : **2663**
E D S P I C : **739**

UNIVERSITE BLAISE PASCAL - CLERMONT II
ECOLE DOCTORALE SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND
&
UNIVERSITE DES SCIENCES, DES TECHNIQUES ET DES TECHNOLOGIES DE BAMAKO

Thèse

Présentée par

Hamidou TOGO

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

SPECIALITE : INFORMATIQUE

PARALLELISATION DE SIMULATEUR DEVS PAR METAMODELISATION ET TRANSFORMATION DE MODELE

Soutenue publiquement le 23 Décembre 2015 devant le jury :

Pr. Moussa LO	Président et examinateur
Pr. Pierre SIRON	Rapporteur et examinateur
Pr. Hans VANGHELUWE	Rapporteur et examinateur
Pr. Mamadou Kaba TRAORE	Directeur de thèse
Pr. Ouaténi DIALLO	Directeur de thèse
Dr. Jacqueline KONATE	Examinateur

SOMMAIRE

LISTE DES FIGURES

LISTE DES SIGLES

AGL	Ateliers de Génie Logiciel
ALSP	Aggregate Level Simulation Protocol
ATL	ATLAS Transformation Language
BD	Base de Données
CIM	Computation Independent Model
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Models
DEVS	Discrete Event System Specification
DIS	Distributed Interactive Simulation
DoD	Department of Defense
DOHS	Distributed Optimistic Hierarchical Simulation
DSL	Domain Specific Language
DSS	Data Distributed Service
EIC	External Input Coupling
EOC	External Output Coupling
GMP	Generic Model Partitioning
GVT	Global Virtual Time
HLA	High Level Architecture
HTTP	HyperText Transfer Protocol
HIPART	Hierarchical Partitioning
IC	Internal Coupling
IDM	Ingénierie Dirigée par les Modèles
LP	Logical Process
LVT	Local Virtual Time
M&S	Modélisation et Simulation
MD3SEA	Model Driven Distributed DEVS Simulation Engineering Architecture
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MOF	Meta-Object Facility
OMG	Object Management Group
P-DEVS	Parallel Discrete Event System Specification
P2P	Peer to Peer
PDM	Platform Description Model
PIM	Platform Independent Models
PSM	Platform Specific Models
QVT	Query View Transformation
RMI	Remote Method Invocation
RTI	Run Time Infrastructure
SGF	Simulation Graph Framework
SIMNET	Simulator Networking
SPD	Simulation Parallèle et Distribuée
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language
XML	Extensible Markup Language

RESUME

Cette thèse propose une approche d'ingénierie consistant à paralléliser des simulateurs DEVS existants, sans être obligé de modifier les algorithmes de l'implémentation initiale, mais en injectant des composants additionnels adaptés au protocole de communication inter-composants en vigueur. Les algorithmes de simulation de ces nouveaux composants appelés « Manteaux », sont définis.

Une démarche d'ingénierie permettant de systématiser le passage d'une implémentation à sa contrepartie parallèle et distribuée est ensuite proposée. Cette démarche s'appuie sur les principes de méta modélisation et de transformation de modèles inspirés de l'Ingénierie Dirigée par les Modèles (IDM). Sa généricité en garantit la réutilisabilité avec tout simulateur séquentiel DEVS.

Mots clés : DEVS, M&S, Simulation repartie, Méta-modèle, Graphe de simulation, Transformation de modèle.

ABSTRACT

This thesis proposes an engineering approach to parallelize existing DEVS simulators without having to modify the algorithms of the initial implementation, but by injecting additional components suitable for inter-component communication protocol into force. The simulation algorithms of these new components called "Coats" are defined.

An engineering approach to systematize the passage from one implementation to its counterpart parallel and distributed is then proposed. This approach is based on metamodeling and models transformation principles inspired of Model Driven Engineering (MDE). Its genericity guarantees the reusability with any sequential DEVS simulator.

Keywords: DEVS, M&S, Distributed simulation, Meta-model, simulation graph, model transformation.

REMERCIEMENTS

Comment commencer ces remerciements par une autre personne que mon directeur de thèse, Pr. Mamadou Kaba TRAORE (Mams !) de l'Université Blaise Pascal, Clermont-Ferrand 2. Son soutien de tous les jours depuis mon DEA et durant mes années de thèse m'ont permis de mener à bien ces travaux. L'impact qu'il a eu sur mes recherches et sur ma vie personnelle a largement dépassé le cadre de ses fonctions. Je tiens donc tout particulièrement à exprimer ma plus profonde gratitude à Mams pour avoir dirigé ces travaux et m'avoir soutenu dans cette étude. Ses conseils ont été indispensables à la concrétisation de cette recherche.

Je remercie également Pr. Ouaténi DIALLO de l'Université des Sciences, des Techniques et des Technologies de Bamako (USTT-B) pour m'avoir co-encadré. Sa disponibilité, ses critiques constructives m'ont permis de profiter de son expérience et ainsi, de faire aboutir cette thèse.

Je remercie également mes deux rapporteurs, Pierre CIRON de l'Institut Supérieur de l'Aéronautique et de l'Espace de Toulouse en France et Hans VANGHELUWE de l'Université d'Anvers en Belgique, pour la lecture qu'ils ont fait de ce manuscrit et les remarques positives et critiques qui m'ont permis de prendre du recul sur mon travail.

Je tiens aussi à remercier mes examinateurs, Pr. Moussa LO de l'Université Gaston Berger à Saint Louis au Sénégal, président de mon jury et Dr Jacqueline KONATE SOGOBA de l'USTT-B, qui ont bien voulu examiner ce manuscrit.

Je tiens particulièrement à remercier l'ensemble des membres du GReP (Groupe de Recherche du Pôle) pour l'ensemble des échanges que nous avons eu. Je remercie particulièrement Oumar Y. MAIGA, mon compagnon de lutte depuis le DEA à la FAST. De véritables liens d'amitié se sont tissés entre nous. Merci beaucoup Oumar !

En écrivant ces remerciements, je pense évidemment à mes compagnons d'infortune (Adedoyin, Hamzat, Ignace, Youssouf, Hawa) de la salle des thésards du PUTV qui fut autant un lieu de vie qu'un lieu de travail durant ces quelques années passées au pôle.

Je remercie tout le Département d'Enseignement et de Recherche (DER) de mathématiques et d'informatique de la FST/USTT-B en particulier Pr Yaya Koné, chef de DER, pour son suivi dans les démarches administratives ainsi que toute l'administration de la Faculté des Sciences et Techniques (FST) de l'Université de Sciences, des Techniques et Technologie de Bamako (USTTB). Mes remerciements vont également au Pr Françoise Paladian, directrice de l'Ecole Doctorale des Sciences pour l'Ingénieur (EDSPI) de Clermont-Ferrand ainsi qu'à toute son équipe pour leur soutien administratif.

Je remercie la direction du Pôle Universitaire et Technologique de Vichy (PUTV) et l'équipe du LIMOS (Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes) de Clermont-Ferrand pour leur accueil pendant toute la durée de la thèse.

Je remercie également Dr Sinaly DEMDELE, coordinateur du Programme de Formation des Formateurs (PFF) des universités de Bamako, pour sa disponibilité et son accompagnement.

Je termine ces remerciements par une pensée affectueuse à tous mes proches, mes parents, ma famille et mes amis. Je ne les citerai pas ici, bien qu'ils aient une part très importante dans mon parcours pour leur soutien sans faille. Ils ont été également d'un soutien indispensable dans des moments quelque fois difficiles.

Je remercie Salimata, mon épouse, pour son soutien quotidien indéfectible et son enthousiasme contagieux à l'égard de mes travaux comme de la vie en général.

Chapitre I.
INTRODUCTION GENERALE

L'activité de Modélisation et Simulation (M&S) joue un rôle de plus en plus primordial dans l'étude des systèmes (naturels ou artificiels) complexes.

Le but de la modélisation est de construire un modèle afin de résoudre un problème d'analyse ou de conception. Un modèle est une représentation simplifiée du comportement observable et de la structure d'un système réel. La résolution de ces problèmes, appliquée sur le modèle, permet d'obtenir une solution dans le monde du modèle. Cette solution est interprétée dans le monde réel, et devient la solution pour le système réel

Le but de la simulation est de reproduire le comportement d'un modèle à l'aide d'un programme informatique. Elle est principalement utilisée comme alternative à la résolution des modèles mathématiques, dans le cas où il est analytiquement difficile, voire impossible de les résoudre.

DEVS pour *Discrete Event System Specification* [Zeigler 1976] est un paradigme de M&S unificateur en simulation à évènements discrets et est largement reconnu comme tel par la communauté scientifique. Le formalisme DEVS est défini par un cadre de modélisation qui sépare modèle et simulateur et un protocole de simulation qui donne la sémantique opérationnelle du formalisme. Il en existe plusieurs implémentations proposées dans la littérature, et qui sont essentiellement des outils de laboratoire de Recherche. Le travail de cette thèse se situe dans le contexte de M&S avec DEVS.

De nos jours les systèmes à modéliser deviennent de plus en plus complexes, il est donc difficile d'effectuer leur simulation sur un seul processeur physique. L'une des alternatives pour pallier cela, est la simulation répartie (appelée aussi Simulation parallèle et Distribuée - SPD) [Fujimoto 1990]. La SPD permet de répartir la simulation sur plusieurs processeurs physiques afin d'exploiter au maximum leurs puissances de calcul. Cela permet donc de réduire les temps d'exécution. La caractéristique « Distribuée » est utilisée lorsque les processeurs sont géographiquement dispersés, sinon c'est la caractéristique « Parallèle » qui est utilisée. Les problématiques, méthodes et principales techniques restent communes dans les deux cas.

La maturité de la SPD reste théorique, sa systématisation aux formalismes de M&S (tels les Réseaux de Pétri, les Réseaux de File d'Attente, les Automates Cellulaires, DEVS…), recèle des difficultés qui demandent encore des efforts de Recherche et d'Ingénierie non négligeables. Dans le cas de DEVS, les coûts en temps et en efforts restent élevés pour implémenter des plateformes de SPD, en particulier lorsqu'il s'agit de paralléliser des implémentations DEVS séquentielles existantes.

Il existe des implémentations distribuées de DEVS dans la littérature, mais elles redéfinissent l'architecture et surtout les algorithmes initiaux de simulation. Par conséquent, elles sont obtenues au prix d'une réingénierie aussi coûteuse, voire beaucoup plus coûteuse, que l'ingénierie de la solution séquentielle de départ.

Nous proposons, dans cette thèse, une approche d'ingénierie consistant à paralléliser des simulateurs DEVS existants, sans être obligé de modifier les algorithmes de l'implémentation initiale, mais en injectant des composants additionnels adaptés au protocole de communication inter-composants en vigueur. Nous appelons Manteaux ces nouveaux composants, dont nous définissons d'abord les algorithmes de simulation. Puis nous proposons une démarche d'ingénierie permettant de systématiser le passage d'une

implémentation DEVS séquentielle à sa contrepartie parallèle et distribuée. Cette démarche s'appuie sur les principes de méta modélisation et de transformation de modèles inspirés de l'Ingénierie Dirigée par les Modèles (IDM). Sa généricité en garantit la réutilisabilité avec tout simulateur séquentiel DEVS.

Ce document est organisé en 5 chapitres, dont cette introduction qui en constitue le chapitre 1.

Le chapitre 2 porte sur l'état de l'art relatif aux travaux présentés dans cette thèse. Comme notre travail se situe à la frontière des trois domaines distincts que sont le champ de la Simulation répartie, le paradigme DEVS, et l'Ingénierie Dirigée par les Modèles, ce chapitre visite chacun de ces domaines. D'abord, les concepts et principes des approches de simulation répartie sont survolés, avec une distinction entre approches dites pessimistes ou conservatives et approches dites optimistes. Les grands standards de simulation répartie (tels DIS, HLA…) sont aussi présentés. Puis, nous présentons le formalisme DEVS, à travers sa syntaxe et sa sémantique opérationnelle. Cette dernière introduit la notion d'arbre de simulation, concept central aux travaux de cette thèse. Les travaux existants relatifs à la parallélisation de DEVS sont également présentés. Enfin, l'approche d'Ingénierie Dirigée par les Modèles est abordée. L'inspiration principale de cette thèse en la matière est l'architecture MDA, dont les concepts et principes sont rappelés, avec un accent particulier apporté à la méta-modélisation et à la transformation de modèles.

Le chapitre 3 présente la première des trois principales contributions de cette thèse, à savoir la définition de composants dont l'ajout à une implémentation séquentielle permet d'en obtenir une version parallèle, sans pour autant engendrer la modification du code des composants initiaux existants. L'architecture globale du nouveau schéma de simulation est montrée, et l'algorithme permettant de simuler ces nouveaux composants, appelés manteaux, est donné. Une étude des performances de cette nouvelle approche est faite sur un cas d'étude de simulation de trafic urbain. Les expérimentations sont faites sur un réseau de PCs identiques.

Le chapitre 4 propose les deux autres contributions principales de la thèse. L'une d'elles est une approche inspirée de l'Ingénierie Dirigée par les Modèles pour systématiser le passage d'une version séquentielle DEVS à sa contrepartie parallèle. Dans le contexte de DEVS, ce passage s'interprète comme la transformation d'un arbre de simulation en graphe de simulation. Pour le modélisateur, il s'agira d'appliquer des calques au modèle qu'il a conçu, pour générer des simulateurs parallèles capables de prendre en charge son modèle. Cette démarche fait donc remonter au niveau de la modélisation, des décisions de répartition qui sont traditionnellement traitées au niveau de l'implémentation. L'autre des contributions est la parallélisation, par injection de manteaux, de SimStudio, notre package de simulation séquentiel DEVS initialement développé dans notre Laboratoire de Recherche quelques années plus tôt.

Enfin, le chapitre 5 permet de conclure et de présenter les perspectives futures envisagées pour ces travaux.

Chapitre II.
ETAT DE L'ART

1. Introduction

Ce chapitre visite les 3 grands domaines, à l'intersection desquels s'articule cette thèse.

La section 2 aborde le champ de la Simulation répartie, et en présente les concepts, les techniques et les protocoles. Cette présentation est indépendante à la fois des formalismes utilisés pour exprimer les modèles de simulation et des langages utilisés pour implémenter ces modèles. Il est donc important de comprendre que l'application des démarches de simulation répartie nécessite leur adaptation à la sémantique des formalismes de modélisation pour simulation utilisés.

La section 3 présente la syntaxe et la sémantique opérationnelle du formalisme DEVS, qui est le formalisme de simulation à événements discrets dans lequel se situent nos efforts. La compréhension du paradigme de simulation qui se rattache à l'usage de ce formalisme à caractère universel est essentielle à une bonne application des techniques réparties vues dans la section précédente.

La section 4 présente l'Ingénierie Dirigée par les Modèles, avec un focus sur l'architecture MDA et les concepts de méta modélisation et de transformation de modèles. Cette approche d'ingénierie logicielle préconise, pour systématiser le passage du niveau conceptuel des problèmes à l'implémentation de leurs solutions, que le traitement de toutes les questions liées à ce passage soit ramené à de la manipulation de modèles. Comme notre objectif est de rendre aisé le passage de simulateurs DEVS séquentiels à leurs contreparties parallèles, cette démarche de systématisation nous semble toute indiquée.

2. Simulation Répartie

En dehors de l'approche expérimentale, la résolution de problèmes passe par deux autres types d'approche :

- Les méthodes analytiques, qui recherchent une solution mathématique ; ces méthodes ne sont pas toujours applicables, en particulier sur certains modèles complexes.
- Les méthodes basées sur la simulation informatique (aussi appelée Modélisation et Simulation ou M&S), qui traitent des modèles plus complexes; elles permettent de résoudre les problèmes auxquels les méthodes analytiques ne peuvent pas répondre (du fait de la complexité du modèle).

Notre travail se situe dans le cadre de la M&S.

2.1. Modélisation et Simulation

La simulation informatique désigne un procédé selon lequel on exécute un programme informatique sur un ordinateur en vue de simuler un phénomène réel complexe. Cette approche trouve toute son importance dans l'ingénierie des systèmes complexes pour lesquels il est difficile d'avoir une vision globale, même si l'on est capable d'appréhender chacun des composants pris individuellement. Cela permet donc d'étudier le fonctionnement et les propriétés du système modélisé ainsi qu'à en prédire son évolution. D'où son utilisation

généralisée, de nos jours, à tous les domaines (industrie, aéronautique, militaire, médecine…). Ses principaux intérêts sont :

- La réduction des coûts de conception des systèmes. Elle permet de mieux appréhender la complexité de systèmes ainsi que d'optimiser les performances et/ou la fiabilité avant la phase de conception réelle du système.
- La vérification du comportement des systèmes et la validation de leur modèle de conception.
- L'accessibilité par des utilisateurs qui ne sont pas forcément des experts de la simulation, d'analyser et d'appréhender le comportement du système simulé, par usage d'environnements riches de M&S.
- L'applicabilité, même lorsqu'aucune autre solution n'est utilisable ou envisageable (par exemple, suite à l'interdiction d'effectuer des essais nucléaires réels, le Commissariat à l'Energie Atomique en France a recours à des simulations pour poursuivre ses expériences).

Les tous premiers algorithmes de simulation qui ont été développés (par la communauté) sont adaptés au cas où le modèle concret est un programme monolithique [Leroudier 1980]. Puis sont apparues des techniques basées sur une partition du programme de simulation en unités logicielles ayant, chacune, une cohérence interne, une temporalité propre, et une autonomie relative vis-à-vis du reste des autres unités, et pouvant s'exécuter sur des hôtes matériels différents. Cette autonomie relative implique des besoins d'échange et de synchronisation entre unités logicielles. Ceci a donné naissance au champ de la simulation répartie (dite aussi Simulation Parallèle et Distribuée -SPD), l'objectif étant d'utiliser au maximum la puissance des ordinateurs (mono ou multiprocesseurs), de pouvoir travailler sur des ordinateurs distants et/ou de réutiliser des simulations existantes en les interconnectant.

C'est dans ce contexte que se situe notre travail.

2.2. De la simulation séquentielle à la simulation répartie

Les approches de simulation répartie permettent de distribuer un simulateur sur plusieurs nœuds appelés Processus Logiques (LP pour Logical Process) et de les exécuter de manière concurrente. Ces LPs gèrent, chacun, une horloge locale dont la date est appelée LVT (pour Local Virtual Time), traitent les événements de simulation qui apparaissent à leur niveau selon leurs propres algorithmes (le LVT étant toujours avancé à la date d'occurrence de l'événement en cours de traitement), et sont localisés sur des processeurs différents, géographiquement distants (simulation distribuée) ou non (simulation parallèle). Ceci nécessite d'introduire des algorithmes supplémentaires de synchronisation entre les LPs pour que les communications qu'ils doivent nécessairement entretenir, afin de traiter les dépendances entre événements qu'ils gèrent, s'appuient sur les mêmes références temporelles.

Une simulation (répartie ou non) doit toujours respecter le *principe de causalité*, qui dit que le futur ne doit pas influencer le passé (sinon, on parle d'*erreur de causalité*). Il est démontré dans [Fujimoto 2000] que ce principe est respecté dans le cadre d'une simulation répartie si chaque LP respecte la contrainte de causalité locale et que les interactions entre LPs se font uniquement par envois de messages estampillés par leurs dates d'émission. La *contrainte de causalité locale* impose que chaque LP traite ses évènements dans l'ordre croissant de leurs dates d'occurrence.

Pour respecter le principe de causalité, deux types de synchronisations entre les nœuds sont possibles : une synchronisation préventive qui évite toute situation d'erreur de causalité, et une synchronisation curative qui ne se soucie pas de la survenue d'une erreur de causalité mais est capable de la détecter et de la corriger. Ainsi sont nées les approches « pessimistes » [Misra 1986] (synchronisation préventive) et « optimistes » [Jefferson 1985] (synchronisation curative).

2.3. Approches pessimistes

Dans le cas pessimiste, un LP ne traite un événement dont la date d'occurrence est t, que lorsqu'il est sûr de ne pas recevoir des autres LPs de message d'estampille t' < t, ce afin de respecter la contrainte de causalité locale. Cette surveillance impose à un LP d'affecter une file de réception spécifique à chaque autre LP susceptible de lui envoyer un message (comme le montre la Figure 1), et de s'assurer qu'aucun traitement d'événement ne sera fait tant qu'au moins une de ces files est vide.

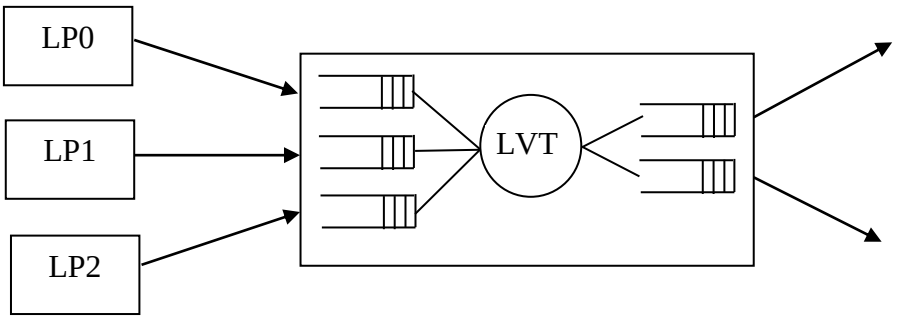


Figure 1. Processus Logique

2.3.1. Problématique de l'inter blocage

Le LP se bloque donc, si une de ses files de réception de message est vide, car il ignore alors la date du prochain message qui arriverait dans cette file et ne peut donc garantir que ce prochain message sera d'estampille supérieure ou inférieure à celui des autres messages en attente dans ses autres files.

Le problème de cette approche est la possibilité d'une situation d'inter-blocage des LPs lorsque ces derniers sont reliés en boucle et s'influencent en chaîne fermée. Si l'un des LPs se bloque, il entraîne le blocage de tous les autres.

La Figure 2 illustre cette situation d'inter-blocage. Par exemple, LP1 après réception d'un message de LP0, émet un message soit vers LP2 soit vers LP3 en fonction d'une certaine règle. Si au cours de la simulation aucun message n'est envoyé vers LP2, la file de réception de LP4 nourrie par LP2 sera en famine et finira par se vider, et LP4 ne pourra plus faire avancer son temps de simulation. Il sera donc bloqué indéfiniment en attendant l'arrivée d'un message sur la file venant de LP2, bien qu'il ait des messages en attente en provenance de LP3. Ainsi LP0 à son tour sera aussi bloqué en attendant l'arrivée de message provenant de LP4. Ceci finira par bloquer LP1, qui à son tour affamera LP3 et LP2.

Il faut noter que cette situation d'inter-blocage ne correspond pas à un inter-blocage du système réel, mais résulte de la mise en œuvre du schéma d'exécution distribué destiné à garantir le principe de causalité.

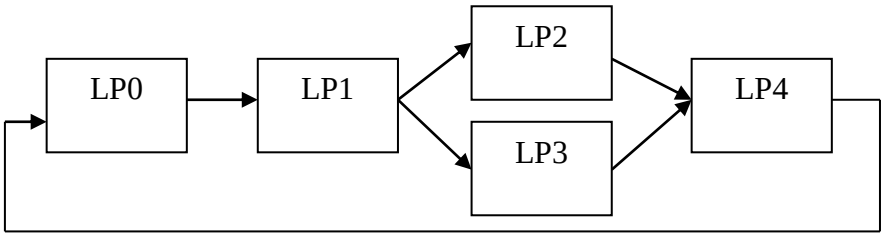


Figure 2. Situation pouvant conduire à un inter-blocage

Pour faire face à cette situation d'inter-blocage, la littérature propose deux classes de solutions : (1) la prévention de l'inter blocage, ou (2) la détection et la guérison de l'inter blocage. La plus emblématique des solutions de la première classe est celle dite à messages nuls [Chandy & Misra 1979], [Bryant 1977]. Celle des solutions de la seconde classe est celle dite du plus petit LVT [Chandy & Misra 1981].

2.3.2. Approche à messages nuls

La solution avec message nul a été proposée dans [Chandy & Misra 1979], [Bryant 1977]. L'idée de base de cette solution est de faire émettre par les processus, en plus des messages normaux de la simulation, des messages de contrôle appelés messages nuls.

Un message nul n'a d'autre contenu que son estampille (date d'occurrence). Lorsqu'un LP transmet un message daté sur une de ses sorties, il transmet également un message nul de même date sur toutes ses autres sorties. Le traitement de ces messages par les LPs récepteurs n'implique aucun autre calcul que celui permettant de faire évoluer le LVT. En effet, un LP inspecte chacune de ses files de réception et prélève le message d'estampille minimal dans l'une d'elles (les messages étant rangés par estampille croissante dans les files, seul le message en tête de chaque file est inspecté). Si ce message n'est pas nul, il engendre un traitement conforme aux règles de la simulation, y compris l'avancée du LVT à la date indiquée par l'estampille de ce message ; si ce message est nul, la seule conséquence est l'avancée du LVT à la valeur indiquée par l'estampille du message nul.

Sur l'exemple précédent (Figure 2), LP1 émet un message nul vers LP2 (respectivement LP3) à chaque fois qu'il émet un message vers LP3 (respectivement LP2). Ainsi LP4 recevra régulièrement des messages sur chacune de ses files d'entrée et pourra donc faire progresser son temps de simulation.

Cette solution ne suffit pas à prévenir l'inter-blocage dans tous les cas. Prenons l'exemple de la Figure 3 où une boucle de rétroaction directe existe entre LP3 et LP2. Au temps 5, LP3 émet le message m1 vers LP4, il émet donc également un message nul de même date vers LP2. Au temps 10, LP1 émet le message m2 vers LP2. Les deux files d'entrée de LP2 contiennent, chacune, un message, LP2 peut donc prélever (NULL, 5) et faire avancer son LVT à 5. Mais la consommation du message nul ne provoque l'émission d'aucun message vers LP3, et le système se bloque.

Pour résoudre ce problème il faut connaitre une information supplémentaire appelée *Lookahead*. Dans la situation de la Figure 3 par exemple, LP2 sait qu'il ne recevra aucun message ayant une date strictement inférieure à 5. Il peut en déduire qu'il n'enverra pas de message avant la date 5+ε (ε > 0), où ε représente la durée minimale de traitement d'un événement par LP2. Pour LP2, cette valeur ε constitue une certaine visibilité sur le futur qui est appelée son Lookahead.

Ainsi, de manière générale, dans le cas où un LP n'a aucune sortie à faire, il envoie un message nul daté avec son Lookahead, i.e. avec la date au plus tôt du prochain envoi de message normal. Le Lookahead est fortement dépendant de la nature du modèle à simuler.

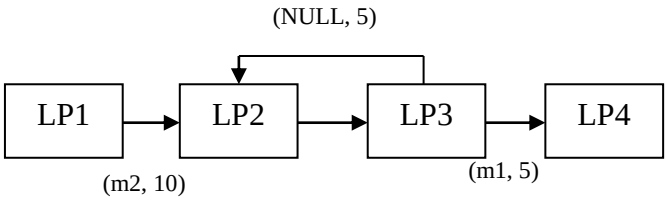


Figure 3. Inter blocage dans l'approche à messages nuls

2.3.3. Approche du plus petit LVT

Cette stratégie a été introduite dans [Chandy & Misra 1981]. Dans ce cas la simulation consiste à répéter la séquence suivante:

(i) Simuler jusqu'à l'inter-blocage.
(ii) Détecter l'inter-blocage.
(iii) Guérir l'inter-blocage en relançant l'exécution d'un ou plusieurs LPs.

Dans la première phase, les LP n'émettent que des messages de simulation. Ils font progresser la simulation en itérant sur la consommation du message reçu sur la file d'entrée ayant la plus petite date. Quand la file ayant la plus petite date est vide le LP se bloque.

Un processus de contrôle détecte alors l'inter-blocage grâce un algorithme (comme celui de [Dijkstra & Scholten 1980] ou celui de [Chandy et al.1983]). Quand le processus de contrôle a détecté l'inter-blocage, il demande aux LPs de démarrer un calcul réparti qui permet de déterminer le, ou les LPs pouvant redémarrer sans introduire de violation du principe de causalité. Il faut noter qu'il existe toujours au moins un LP pouvant reprendre la simulation. En effet, parmi tous les messages en attente dans les files d'attente de tous les LPs, celui qui a la plus petite estampille T peut être consommé par son destinataire car plus rien ne peut modifier l'état du système avant la date T. L'algorithme de guérison consiste alors à calculer cette valeur T. Quelques améliorations de cet algorithme de guérison ont été développées (voir [Chandy & Misra 1981] et [Fujimoto 1990]).

2.4. Approches optimistes

Alors que les approches pessimistes évitent toute violation de la contrainte de causalité locale, les approches optimistes l'autorisent et prévoient un mécanisme de réparation. Ainsi, chaque LP traite les messages au fil de leur réception, sans se préoccuper de savoir si toutes ses files de réception sont non vides. Il est donc possible qu'il reçoive ultérieurement, sur une de ses

files vides, un message dont l'estampille sera inférieure à son LVT, provoquant ainsi une erreur de causalité. Ce message est appelé *trainard*, et le LP doit alors revenir à un passé antérieur à l'estampille du trainard pour pouvoir le traiter correctement. Ce retour au passé est appelé *Roll back*.

Beaucoup de mécanismes optimistes ont été proposés dans la littérature, mais le *Time Warp* [Jefferson 1985] en est le fondateur.

Lorsqu'un LP traite un message dont l'estampille T est inférieure au LVT, il annule tous les messages qu'il a déjà expédiés avec une estampille supérieure à T (appelons-les « mauvais messages »), et restaure le LP dans l'état antérieur à T le plus proche du LVT. Pour cela, il doit gérer un historique de ses états successifs, y compris ses files de message. L'annulation des mauvais messages consiste à demander aux LPs récepteurs de ces mauvais messages, de les annuler de leurs files de messages reçus. Pour ce faire, le LP envoie des anti-messages, qui possèdent exactement les mêmes caractéristiques que les mauvais messages, à l'exception d'un drapeau signalant leur caractère d'anti-message.

Un anti-message lorsqu'il est reçu par un LP, provoque l'annulation du message normal correspondant si ce dernier est encore présent dans la file des messages reçus, sinon provoque à son tour un Rollback de ce LP. Cette stratégie nécessite donc de conserver les états successifs des variables de chaque LP, y compris la liste de tous les messages émis et reçus.

En fait, le Rollback peut se résumer ainsi :

- Défaire une action locale consiste simplement à revenir à un ancien état des variables locales du LP, ancien état que l'on aura sauvegardé.
- Une émission de message sera défaite en émettant un anti-message vers le même destinataire, avec la même estampille que le message initial. La réception d'un anti-message provoquant chez le récepteur soit l'élimination du message initial s'il n'avait pas encore été consommé, soit un Rollback jusqu'à la date correspondant à l'estampille de cet anti-message.
- Les actions définitives (i.e., celles sur lesquelles on ne peut pas revenir, comme les entrées/sorties), sont différées jusqu'à ce que la simulation ait progressé au point où on est sûr de ne pas avoir à les défaire.

L'algorithme du Time Warp est composé de deux parties : (1) un mécanisme de contrôle local (implémenté sur chaque LP), et (2) un mécanisme de contrôle global.

2.4.1. Contrôle local

Les opérations de Rollback sont déclenchées par la réception soit d'un message retardataire ou d'un anti-message. La Figure 4 montre que le LP a exécuté les évènements avec les temps de réception 12, 21 et 35. En conséquence, le LVT de ce LP est 35. Ensuite, un traînard arrive à la date 18, ce qui entraine un Rollback sur ce LP.

A la réception du trainard, les actions suivantes sont effectuées (reflétées par la Figure 5) :

- Insertion du traînard dans la file d'entrée (Input Queue) ;
- Annulation des évènements E(21) et E(35) ;
- Restauration de l'état courant du LP à la date antérieure la plus proche de 18 (i.e., 12) ;

- Suppression de tous les états sauvegardés dans l'historique des états successifs du LP (State Queue) dont les dates sont supérieures à 18 ;
- Remise du LVT à 12 ;
- Envoi des anti-messages correspondants à tous les messages envoyés avec une estampille supérieure ou égale à 18.

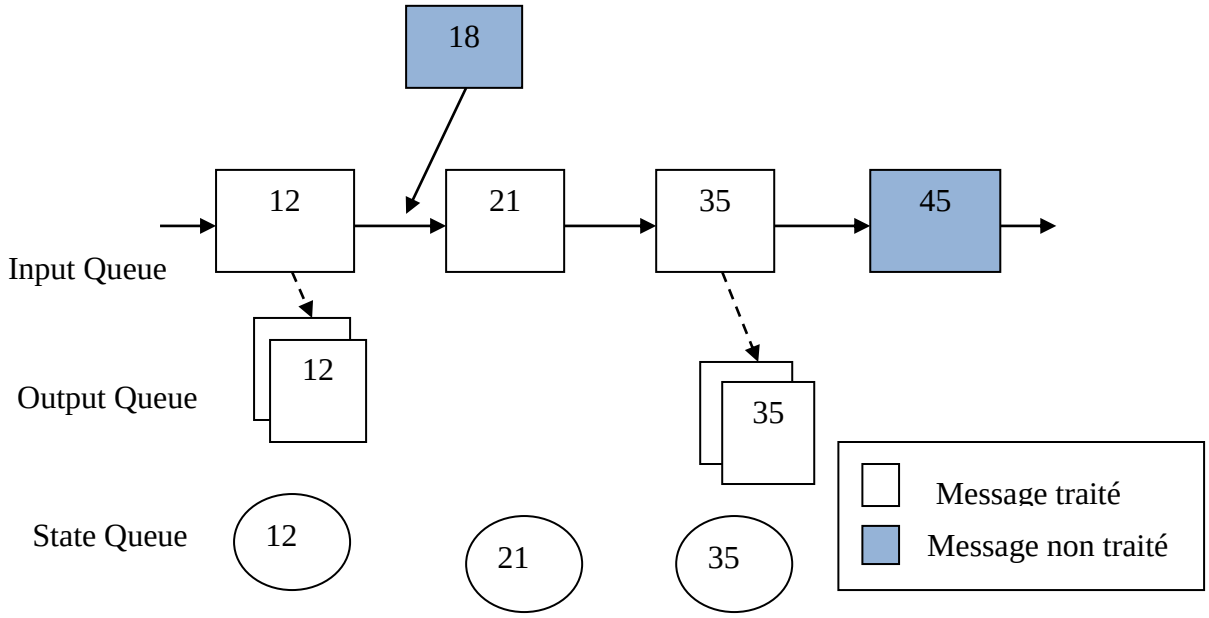


Figure 4. Réception d'un traînard

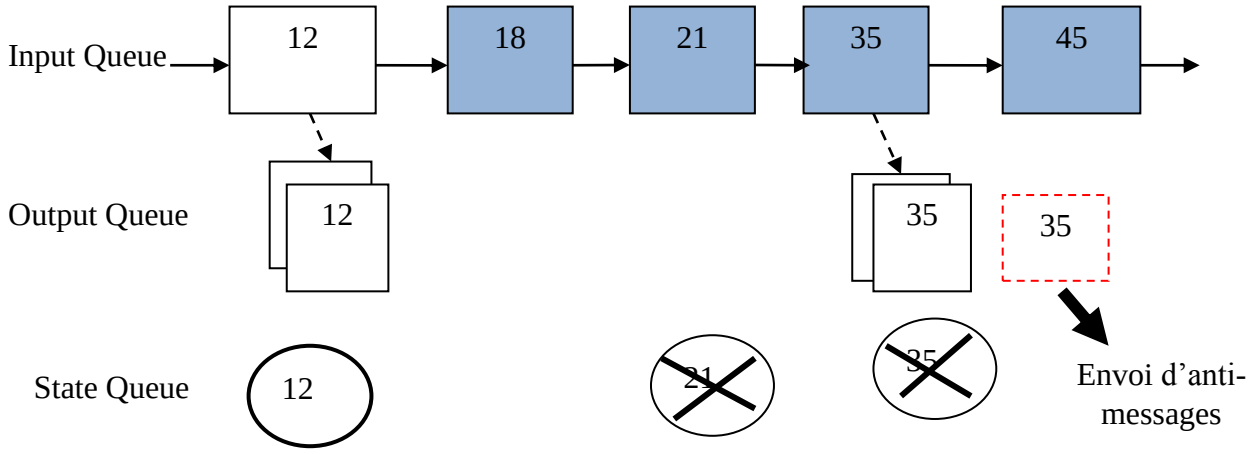


Figure 5. Annulation des actions effectuées et envoi d'anti-message

A la réception d'un anti-message, les deux cas de figure qui se présentent sont décrits par les Figure 6, Figure 7 et Figure 8 :

- Si le message correspondant a été traité alors il est annulé (Figure 6), et le LP effectue un Rollback à la date de l'événement qui précède ce message (Figure 7).
- Si le message correspondant n'a pas encore été traité alors le message et l'anti-message s'annulent tout simplement (Figure 8).
- Si le message correspondant n'était pas encore arrivé au destinataire, alors l'anti-message est placé dans la file de réception des messages, dans l'attente de la réception. Et de la neutralisation du message.

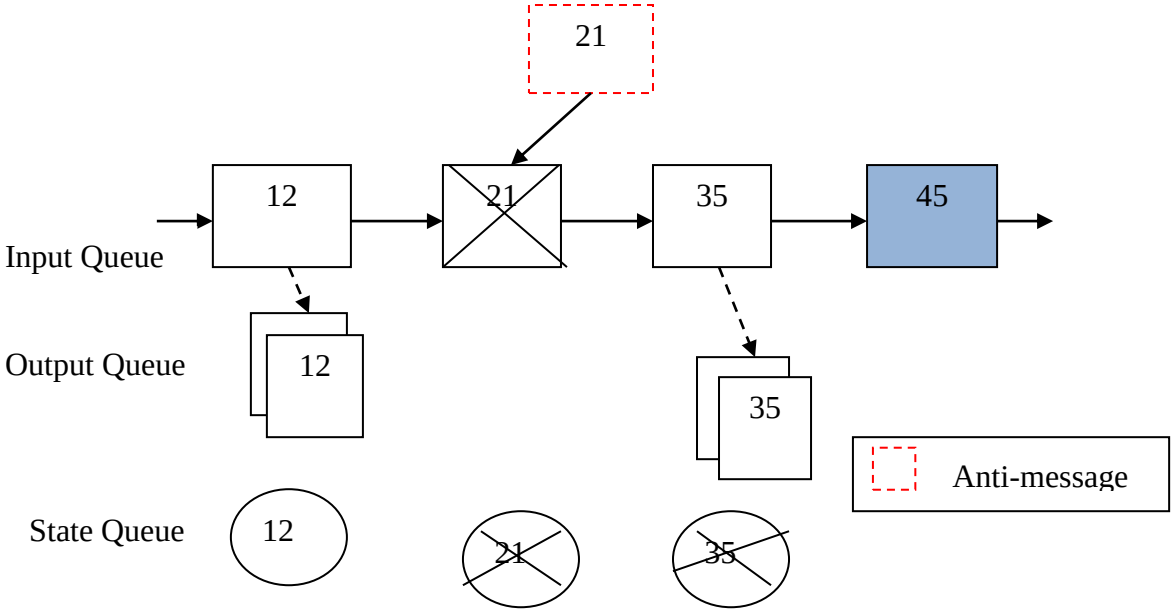


Figure 6. Réception d'anti-message, dont le message a été traité

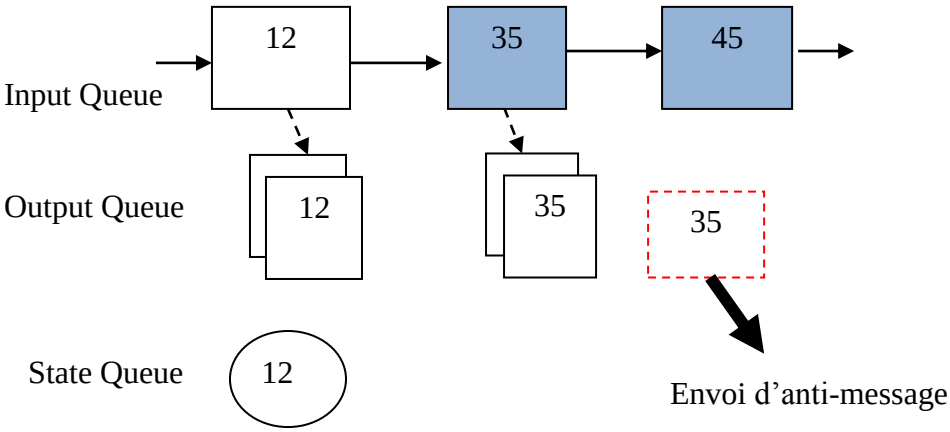


Figure 7. Rollback après réception d'un anti-message

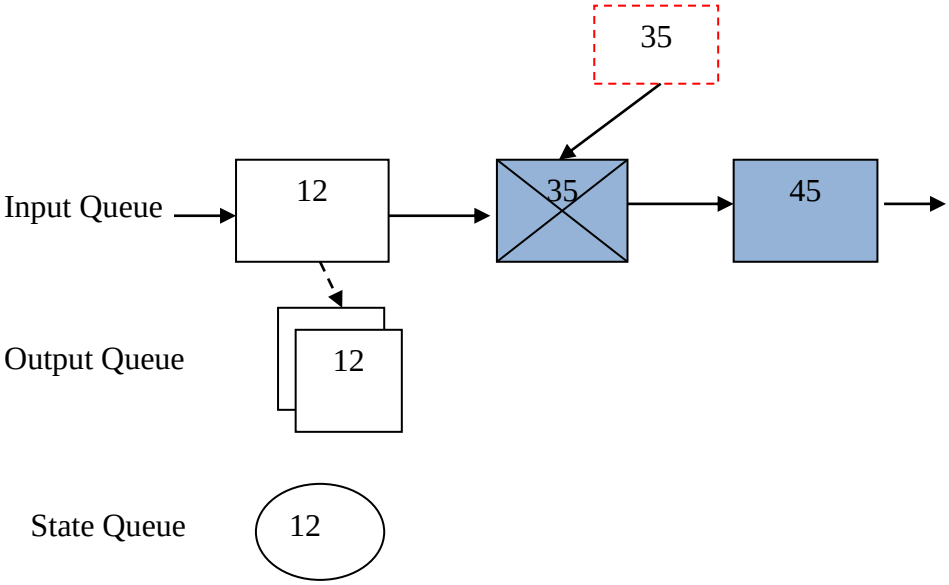


Figure 8. Réception d'anti-message, dont le message n'est pas traité

2.4.2. Contrôle global

Le mécanisme de contrôle global repose sur la notion de Temps Virtuel Global (GVT pour Global Virtual Time). A tout instant de la simulation, le GVT représente une borne inférieure sur les dates de Rollback possibles. Ainsi, toutes les actions effectuées à une date simulée inférieure au GVT ne seront jamais défaites. Les sauvegardes des états ou messages ayant une date inférieure au GVT peuvent être oubliées (libérant ainsi de l'espace mémoire, action appelée *collecte de fossiles*) ; de même, les entrées/sorties dont la date est antérieure au GVT peuvent être réellement réalisées en toute confiance.

Certaines variantes du Time Warp calculent le GVT de manière périodique, d'autres seulement lorsque la simulation manque de mémoire.

2.5. Protocoles de simulation répartie

Dans la mise au point d'une solution de SPD, le protocole de transmission des messages entre LP revêt une importance de premier ordre. En effet, comme les LPs ont vocation à s'exécuter sur des processeurs différents, géographiquement distribués ou non, l'échange de message entre LP repose sur les mécanismes de communication inter processeurs.

Plusieurs protocoles de SPD ont été développés par la communauté scientifique. Elles se répartissent entre les 3 stratégies suivantes :

- Point-à-Point (Peer-to-Peer, ou P2P),
- Client/Serveur, ou
- Producteur/Consommateur (forme plus large du Client/Serveur).

2.5.1. Protocole P2P

Les technologies de SPD en mode P2P les plus visibles sont :

- SIMNET (SIMulator NETworking) qui a été développé et utilisé par l'armée américaine [Miller & Thorpe 1983]. Son développement a commencé dans les années 80 et a été utilisé pour l'entraînement, avant que les standards successeurs voient le jour dans les années 1990.
- DIS (Distributed Interactive Simulation) qui est un standard de SPD développé au sein du symposium de la communication interactive à l'université de Floride pour améliorer les applications tournantes de SimNET. Ses principales implémentations sont OpenDIS, KDIS, DIS 4.0 et Mik (commerciale).

2.5.2. Protocole Client/Serveur

Dans ce domaine, les efforts les plus notables sont :

- ALSP (Aggregate Level Simulation Protocol) qui a été conçu pour supporter la simulation à évènements discrets et a été implémenté avec succès dans les jeux de combats en ligne [Fujimoto 2000].
- CORBA, développée par l'OMG (Object Management Group), et qui vise à mettre en place une norme d'architectures distribuées ouvertes (et pas pour la SPD uniquement)

permettant de faire communiquer des applications en environnement hétérogène (plusieurs systèmes et plusieurs langages) [OMG 2000].

2.5.3. Protocole Producteur/Consommateur

Les efforts les plus notables ici, sont :

- HLA (High Level Architecture) qui a été développé par le bureau de Modélisation et de Simulation en Défense (DMSO) du Département américain de la Défense (DoD). Dans le protocole HLA, chaque simulateur participant, dit fédéré, interagit avec d'autres fédérés au sein d'une collection de simulateurs dite fédération. Les fédérés communiquent entre eux au sein de la fédération à travers un middleware appelé RTI (Run Time Infrastructure) [Fujimoto 2000]. Quelques implémentations de HLA sont : YaRTI (implémenté en ADA95 dans le domaine de l'Aerospatiale), Portico (implémentée en C++ et Java dans le domaine de la Défense en Australie), GERTICO (implémenté en CORBA dans le domaine de la Défense en Allemagne), CERTI (implémenté en C++ dans le domaine de la Défense en France), RTI-NG (implémenté en C, C++, JAVA, CORBA,... dans le domaine de la Défense aux USA), EODISP (implémenté en Java dans le domaine spatial en Europe), Calytrix (pour le jeu en réseau), etc.
- DDS (Data Distribution Service), standard spécifié par l'OMG dont le rôle est de proposer une technologie évoluée d'échanges de données sur des réseaux allant des systèmes embarqués vers les réseaux à grande distance, en se basant sur une architecture producteur/consommateur. DDS fournit de nouveaux aspects non inclus dans HLA. JacORB-DDS, Poccapsule-DDS, RTI-DDS, CoreDX, et OpenSplite DDS en sont quelques implémentations.
- SOA (Service Oriented Architecture), qui est un protocole orienté services permettant de faire communiquer deux sous-systèmes/applications de manière indépendante des plateformes hôtes utilisées. Elle offre une approche d'interopérabilité plus flexible car elle utilise des protocoles de communication standards ouverts comme XML, HTTP et SOAP pour le mode communication entre le producteur et le consommateur.

3. DEVS

Bernard P. Zeigler a défini au milieu des années 1970 une spécification formelle des modèles de simulation à évènements discrets appelée DEVS (pour Discrete Event System Specification) [Zeigler 1976]. C'est une approche de description modulaire et hiérarchique des systèmes dynamiques, enracinée dans la théorie des systèmes. Il propose une syntaxe abstraite, descriptible au moyen de structures mathématiques, ainsi qu'une sémantique opérationnelle définie par des algorithmes.

La syntaxe abstraite permet de spécifier les modèles DEVS, alors que la sémantique opérationnelle permet de construire les simulateurs DEVS. Cette séparation entre modèles et simulateurs, que montre la Figure 9, permet de faire exécuter par un même simulateur plusieurs modèles DEVS différents, mais aussi dans l'absolu de faire exécuter le même modèle DEVS par plusieurs simulateurs DEVS différents.

Au fil des années, une communauté scientifique s'est structurée autour de DEVS, et plusieurs extensions ont été proposées. Dans le cadre de nos travaux, lorsque nous parlons de DEVS,

nous faisons référence à l'approche originelle de spécification DEVS (dite Classic DEVS), même si notre démarche fonctionne également avec les autres variantes. Ceci est dû au fait que le package de simulation que nous avons paralléliser selon notre approche est une implémentation de Classic DEVS.

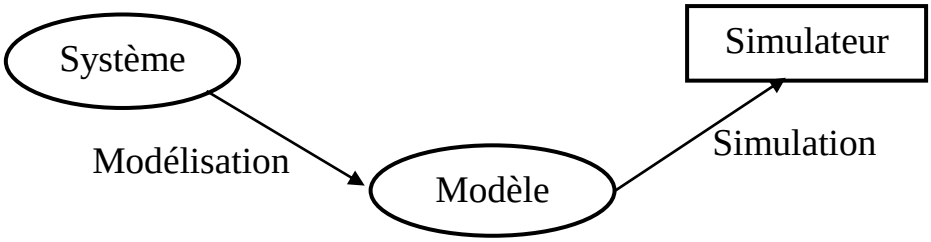


Figure 9. Séparation des préoccupations avec DEVS

3.1. Syntaxe abstraite

DEVS propose de représenter un modèle par une boîte munie d'entrées et des sorties pour interagir avec son environnement (Figure 10). Les événements parviennent au modèle (sous forme de message) par le biais de ports d'entrée. Les réponses du modèle, signalées par des messages émis sur ses ports de sortie, constituent des événements pour l'environnement du système représenté.

Figure 10. Modèle DEVS

Lorsque la structure du modèle est non décomposable, il est dit atomique et son comportement est décrit par des fonctions de transition, de valeur de sortie et d'avancement du temps. Lorsque sa structure est décomposable, il est dit couplé, et son comportement est décrit par l'enchaînement des entrées et sorties des sous-modèles qui le constituent.

3.1.1. Modèle atomique

Un modèle atomique possède un ensemble d'entrées, un ensemble d'états, un ensemble de sorties, une fonction qui détermine les règles de transition interne d'un état du modèle vers un prochain état, une fonction qui détermine les règles de transition externe (c'est à dire due à l'arrivée d'un message en entrée), une fonction qui détermine la valeur de sortie et, une fonction qui détermine la durée de vie d'un état.

La spécification d'un modèle atomique en DEVS est donnée par la structure suivante :

$M = <X, Y, S, \delta int, \delta ext, \delta conf, \lambda, ta>$, où :

- X = {(p, Xp), p ∈ Iport} est l'interface d'entrée, Iport est l'ensemble des ports d'entrée, Xp est l'ensemble des valeurs possibles pour le port p.
- Y = {(q, Yq), q ∈ Oport} est l'interface de sortie, Oport est l'ensemble des ports de sortie, Yq est l'ensemble des valeurs possibles pour le port q.
- S = est l'espace d'état.

- $\delta_{int} : S \rightarrow S$ est la fonction de transition interne.
- $\delta_{ext} : Q \times X \rightarrow S$ est la fonction de transition externe.
- $Q = \{(s,e) / 0 \leq e < ta(s)\}$.
- $\lambda : S \rightarrow Y$ est la fonction de sortie.
- $ta : S \rightarrow IR+$ est la fonction d'avancement du temps.

3.1.2. Modèle couplé

Un modèle couplé décrit une structure par interconnexion de modèles de base, où les modèles de base sont soit des modèles atomiques, soit des modèles couplés. Les échanges entre les modèles se font par des envois et des réceptions d'événements, via les ports d'entrée et de sortie.

La spécification d'un modèle couplé en DEVS est donnée par la structure suivante :

$M = <X, Y, D, \{Md\}_{d \in D}, EIC, EOC, IC, Select>$, où :

- X et Y sont définis comme dans le cas du modèle atomique.
- D est l'ensemble des noms (ou références) des modèles qui composent M.
- M_d est le modèle de nom d.
- $EIC = \{(p, k, d), p \in Iport, k \in Iport_d\}$ est la matrice de couplage des ports d'entrée de M avec les ports d'entrée des M_d.
- $EOC = \{(l, d, q), l \in Oport_d, q \in Oport\}$ est la matrice de couplage des ports de sortie des M_d avec les ports de sortie de M.
- $IC = \{(l, d, k, d'), l \in Oport_d, k \in Iport_{d'}, d \neq d'\}$ est la matrice de couplage des ports de sortie des M_d avec les ports d'entrée des M_d.
- $Select : P(D) - \varnothing \rightarrow D$ est la fonction d'arbitrage.
- P(D) est l'ensemble des parties de D.

Comme le montre la Figure 11, les couplages EIC représentent les couplages entre les entrées du modèle couplé que l'on définit et les entrées des modèles le composant. Les couplages EOC représentent les couplages entre les sorties du modèle couplé que l'on définit et les sorties des modèles le composant. Les couplages IC sont les couplages entre les entrées et les sorties des modèles composant le modèle couplé que l'on définit, sachant qu'un modèle ne peut pas être couplé à lui-même.

La fonction Select est sollicitée lorsque plusieurs modèles composant le modèle couplé doivent, de manière concurrente, effectuer des actions à la même date. Le modeleur doit donc prévoir cette situation, et préciser dans ce cas, lequel des modèles a la priorité (Classic DEVS ayant été conçu sur la base d'une exécution séquentielle).

Il est à noter que SimStudio implémente également une version séquentielle de P-DEVS (pour Parallel DEVS) [Chow 1996], une variante de DEVS qui proscrit la fonction Select, mais introduit une fonction de transition confluente au niveau des modèles atomiques pour traiter les événements simultanés de réception et d'envoi de message. P-DEVS n'est pas une parallélisation de Classic DEVS, mais une manière de gérer la concurrence au niveau conceptuelle. Notre démarche dans ce travail de thèse peut s'appliquer aussi bien à Classic DEVS qu'à P-DEVS.

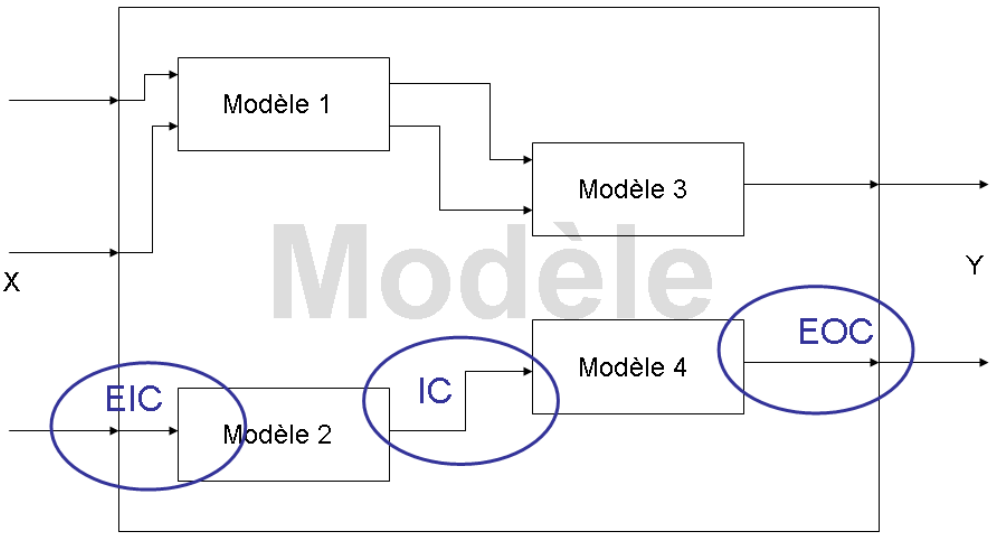


Figure 11. Modèle couplé DEVS

3.2. Sémantique opérationnelle

Un modèle atomique DEVS est supposé être, à tout instant dans un état donné pour une durée de vie donnée (donnée par la fonction d'avancement du temps), à l'issue de laquelle le système change d'état (transition interne) après avoir émis une valeur de sortie. Il est possible qu'une entrée survienne avant la fin de cette durée, provoquant une transition externe (sans émission de valeur de sortie).

Un modèle couplé DEVS achemine les messages émis en sortie de ses composants vers les entrées des autres composants, conformément aux couplages définis par IC. De même, il achemine les messages reçus sur ses propres entrées vers les entrées de ses composants, et ceux émis sur les sorties de ses composants vers ses propres sorties, conformément aux couplages décrits par EIC et EOC.

La sémantique opérationnelle de DEVS est d'abord présentée intuitivement à travers la notion de trajectoires d'entrée, d'état et de sortie, puis plus formellement décrit à travers le protocole de communication établi sur l'architecture générique proposée pour la simulation et les algorithmes associés.

3.2.1. Trajectoires d'entrée, d'état et de sortie

La Figure 12 illustre l'évolution d'un modèle DEVS sur un exemple. A l'état initial t0, le système est dans l'état s0. La fonction ta nous indique que pour l'état s0, le système changera d'état à t0 + ta(s0) si aucun événement extérieur ne survient. A t1 = t + ta(s0), aucune entrée n'a eu lieu. La fonction de sortie est donc activée et Y prend pour valeur la valeur produite par la fonction λ pour l'état s0. Après avoir affecté les ports de sortie, la fonction de transition interne δ_{int} est appliquée. Le système passe dans l'état s1 = δ_{int} (s0) et y restera jusqu'à t1 + ta(s1) sauf en cas d'interruption par un message d'entrée. A l'instant t2, qui est inférieur à t1 + ta(s1), un événement extérieur est placé en entrée. On fait alors appel à la fonction de transition externe δ_{ext} pour déterminer le nouvel état. Dans ce cas, la fonction de sortie n'est pas appliquée. Elle est appliquée exclusivement lors des transitions internes. A l'instant t2, le

système passe dans l'état s2 = δ_{ext} (s1, e, x). Si aucun événement externe n'avait eu lieu, le système serait à t1 + ta(s1) dans l'état s2 = δ_{int}(s1). Et ainsi de suite…

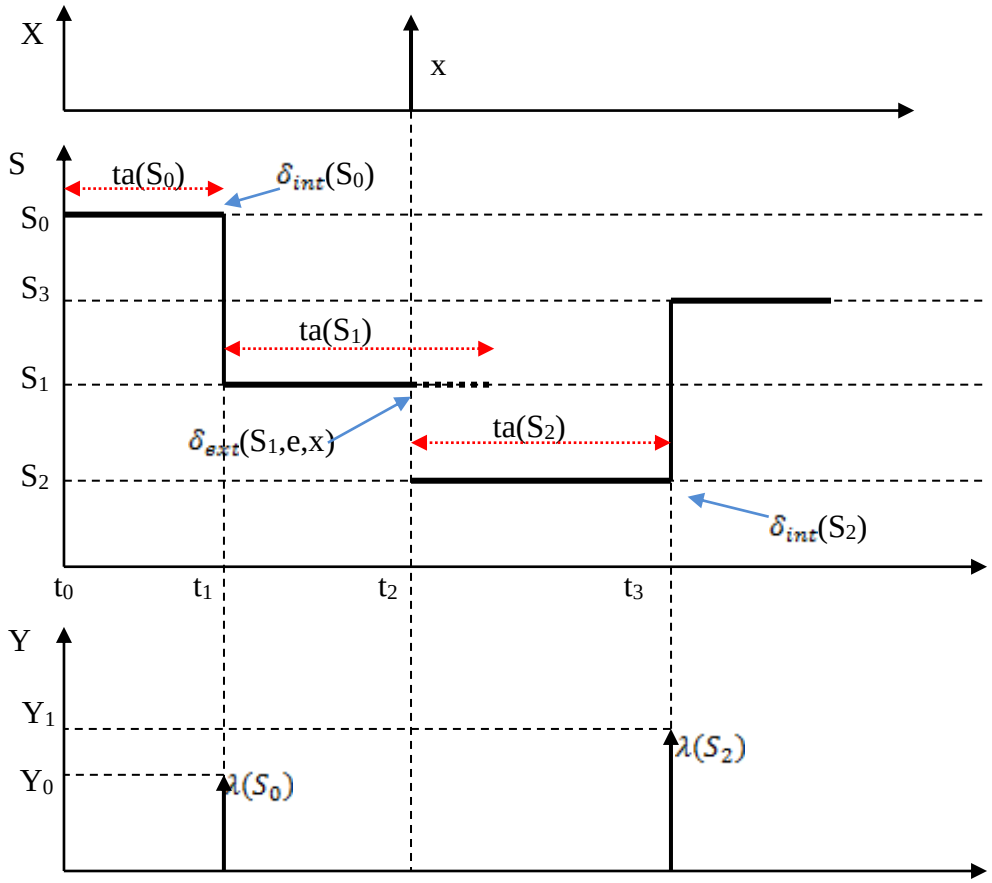


Figure 12. Trajectoires d'entrée, d'état et de sortie

3.2.2. Protocole de communication

Les simulateurs abstraits permettant de simuler les modèles DEVS sont présentés dans [Zeigler et al. 2000]. Un simulateur abstrait représente une description algorithmique permettant de mettre en œuvre les instructions implicites des modèles issus du formalisme DEVS, afin de générer leur comportement. Un tel simulateur est obtenu en construisant à partir du modèle global à simuler, une hiérarchie dont les nœuds sont les composants du modèle qui sont des modèles couplés, et dont les branches sont les composants qui sont des modèles atomiques, puis en faisant correspondre à chaque élément de cet arbre un automate (appelé nœud de simulation). Le nœud de simulation associé à un modèle atomique est appelé simulateur. Celui associé à un modèle couplé est appelé coordinateur. Au sommet de la hiérarchie se trouve un nœud gestionnaire principal appelé coordinateur racine. Un exemple est montré en Figure 13.

La simulation s'effectue grâce à l'envoi de messages spécifiques entre les différents nœuds de l'arbre comme décrit dans la figure ci-dessus. Ainsi plusieurs types de messages sont échangés entre les nœuds :

- Les messages (i, t), qui provoquent à la date simulée t, l'exécution des instructions d'initialisation chez leurs récepteurs.

- Les messages (*, t), qui provoquent une transition interne du modèle associé au nœud récepteur (s'il s'agit d'un simulateur) ou la propagation de ce message vers le nœud fils approprié (s'il s'agit d'un coordinateur).
- Les messages (x, t), qui provoquent une transition externe du modèle associé au nœud récepteur (s'il s'agit d'un simulateur) ou la propagation de ce message vers le nœud fils approprié (s'il s'agit d'un coordinateur).
- Les messages (y, t), qui sont des demandes d'acheminement de message, émises par un nœud vers son parent coordinateur.

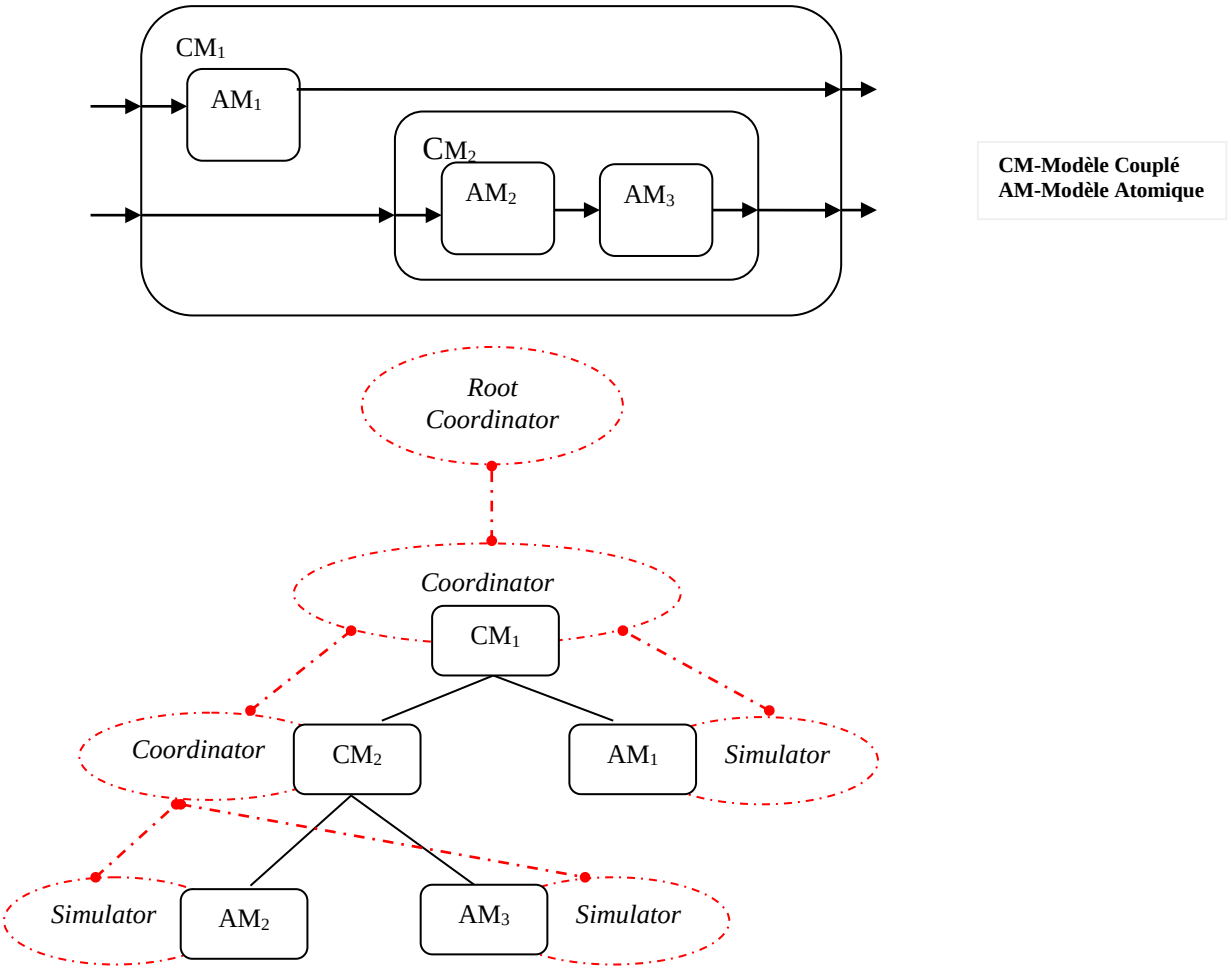


Figure 13. Hiérarchie de modèle et arbre de simulation correspondant

Le comportement dynamique des modèles DEVS est exécuté par les fonctions des nœuds simulateurs associés aux modèles atomiques. Lors de la réception des événements, ils activent les fonctions définies dans le modèle. Ces nœuds manipulent notamment deux variables temporelles nommées tl (temps du dernier évènement) et tn (temps du prochain évènement). Il peut être déduit de ces deux variables la durée de vie de l'état actuel, notée ta(s), où s est l'état actuel du modèle. $tn = tl + ta(s)$. De plus, si l'on connaît la date actuelle globale de la simulation (t), il est possible de calculer le temps écoulé (depuis le dernier événement traité) noté e, par : $e = t - tl$.

La valeur de tn est systématiquement retournée par le simulateur à son coordinateur parent à la fin du traitement d'un événement afin de pouvoir correctement synchroniser les prochains événements à traiter entre les différents simulateurs.

Les Figures 14, 15 et 16 illustrent la dynamique de l'arbre du modèle couplé DEVS, composé de deux modèles, l'un atomique et l'autre couplé, lui-même composé de deux modèles atomiques (Figure 13). Les algorithmes correspondants sont donnés en annexe.

A l'initialisation de la simulation, les simulateurs S0, S1, S2 sont respectivement dans l'état initial St0, St1 et St2. Le coordinateur racine envoie un message d'initialisation, à la date t=t0, à son fils coordinateur, qui le propage à son tour à tous ses fils. Ainsi les simulateurs S0, S1 et S2 calculent leurs temps du dernier évènement tl et leurs temps du prochain évènement tn (Figure 14). Ensuite, les coordinateurs calculent leurs tl et tn en se basant sur les tl et tn des fils (Figure 15).

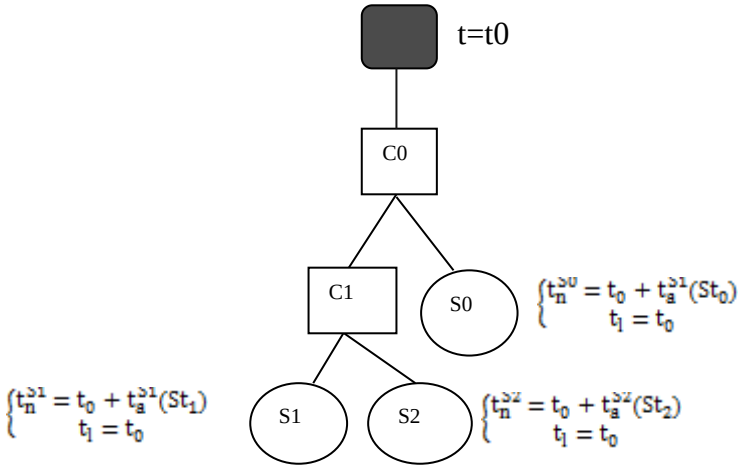


Figure 14. Initialisation des simulateurs

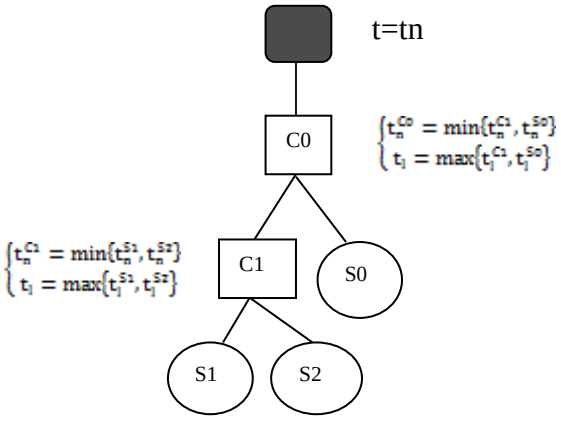


Figure 15. Calcul de tn et tl

Le coordinateur racine avance le temps de simulation à tn (de son fils) et envoie un message (*, t) à son fils qui le propage à ses fils appropriés. Cette action se répète en boucle jusqu'à ce que tn atteigne la date de fin de simulation.

Un simulateur, lorsqu'il reçoit un message (*, t), calcule son message de sortie, l'envoie au coordinateur parent à travers un message (y, t) et effectue sa transition interne. Ce message sera alors acheminé, par le coordinateur parent, en message d'entrée (x, t) pour les destinataires. La réception de message d'entrée (x, t) provoque chez le destinataire le calcul de la valeur de son temps écoulé e, et l'exécution de sa fonction de transition externe (cas de S0, sur la Figure 16).

Après chaque transition (interne ou externe), le temps du dernier évènement tl est mis à la date courante t et le temps du prochain événement est mis à la date t augmentée de la valeur de la fonction d'avancement du temps.

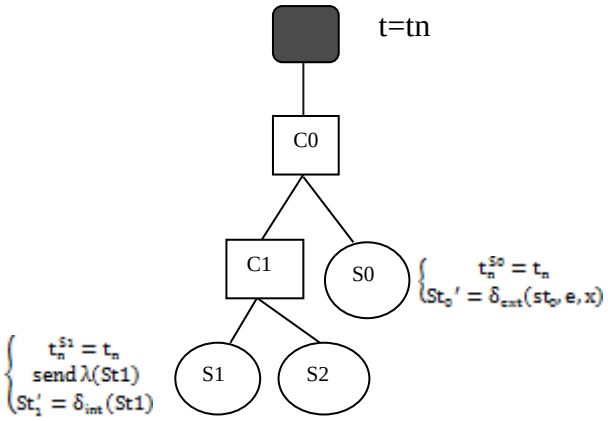


Figure 16. Changement d'état

3.2.2.1. Algorithme du simulateur DEVS

M = <X, Y, S, δint, δext, λ, ta>	// modèle atomique associé
Parent	// coordinateur parent
tl	// date du dernier événement
tn	// date du prochain événement
e	// temps écoulé dans l'état courant
En cas de réception d'un message (i, t)	// initialisation
tl = t- e	
tn = tl + ta(*s*)	
En cas de réception d'un message (*, t)	// transition interne
Si *tn* ≠ *t*, alors Erreur de Synchronisation	
Envoyer *λ*(*s*) sur Y sous forme de message (y, t)	
s = δint(*s*)	
tl = t	
tn = tl + ta(*s*)	
En cas de réception d'un message (x, t)	// transition externe
Si *t < tl* ou *t > tn*, alors Erreur de synchronisation	
e = t ≠ *tl*	
s = δext(*s; e; x*)	
tl = t	
tn = tl + ta(*s*)	

3.2.2.2. Algorithme du coordinateur DEVS

M = <X, Y, D, {Md} $_{d \in D}$, EIC, EOC, IC, Select>	// modèle couplé associé
Parent	// coordinateur parent
tl	// date du dernier événement
tn	// date du prochain événement
En cas de réception d'un message (i, t)	// initialisation

Envoi d'un message (i, t) à chaque fils d de D
tl = max {tl_d}
tn = min {tn_d}
En cas de réception d'un message (*, t) // imminence
Si *tn* ≠ *t*, alors Erreur de synchronisation
Envoi d'un message (*,t) à d* tel que tn_{d*}= *tn* et *d** = *select{d, tn_d = tn}*
tl = t
tn = min {tn_d}
En cas de réception d'un message (x, t) // stimulus extérieur
Si *t < tl* ou *t > tn*, alors Erreur de synchronisation
Envoi d'un message(x, t) aux fils conformément à EIC
En cas de réception d'un message (y, t) // transfert d'envoi
Envoi du même message (y, t) conformément à EOC
Envoi du message sous forme de message (x, t) conformément à IC

3.2.2.3. Algorithme de la racine DEVS

fils	// coordinateur ou simulateur fils
t	// date courante

Envoyer un message (i, t) au fils
t = tn du fils
Tant que la simulation n'est pas finie
Envoyer un message (*, t) au fils
t = tn du fils

3.3. Implémentations DEVS

Comme implémentations majeures de DEVS, citons les suivantes :

- ADEVS [Nutaro 2010] fournit une bibliothèque C++ permettant de construire des simulations à événements discrets basés sur P-DEVS et dynDEVS (extension de DEVS pour les systèmes à structure variable).
- DEVS-C++ [Zeigler et al. 1996] et DEVSJava [Sarjoughian & Zeigler 1998] sont des simulateurs basés sur Classic DEVS implémentés respectivement en C++ et Java.
- GALATEA [Davila & Uzcategui 2000] est une architecture orientée objet pour la modélisation de systèmes multi-agents et leur simulation avec DEVS.
- JAMES [Uhrmacher 2001] est un outil Java de M&S multi-agents basé sur P-DEVS.
- PyDEVS [Delara & Vangheluwe 2002] est une implémentation de DEVS en Python. Par ailleurs, ATOM3 est un environnement de méta-modélisation [Delara & Vangheluwe 2002], dont dérive ATOM3-DEVS, outil pour la modélisation multi-paradigme, la construction de modèles DEVS et la génération de code Python pour le simulateur PyDEVS.
- PowerDEVS [Kofman et al. 2003] est un outil de M&S DEVS des systèmes hybrides. Il est implémenté en C++.
- CD++ [Wainer 2002] est un outil développé en C++ qui implémente Classic DEVS, PDEVS et Cell-DEVS (extension de DEVS pour les automates cellulaires).
- SimStudio [Traore 2008] est le Framework DEVS développé dans notre Laboratoire de Recherche.

Bien d'autres implémentations de DEVS sont proposées, tels SimBeans [Praehofer et al. 1999], SmallDEVS [Janousek et al. 2006], JDEVS [Filippi et al. 2002], DEVS-Scheme [Zeigler & Kim 1993], etc.

3.4. DEVS réparti

Plusieurs approches parallélisation de l'exécution de DEVS ont été proposées dans la littérature. Nous en distinguons deux catégories :

- Les approches que nous appelons projections synchrones, dans lesquelles l'arbre de simulation DEVS est réparti sur plusieurs processeurs (Figure 17) ; et
- Les approches que nous appelons projections asynchrones, dans lesquelles l'arbre de simulation est remplacé par un réseau d'arbres (que nous appellerons plus tard graphe de simulation), chaque arbre de ce réseau étant exécuté sur un processeur distinct (Figure 18).

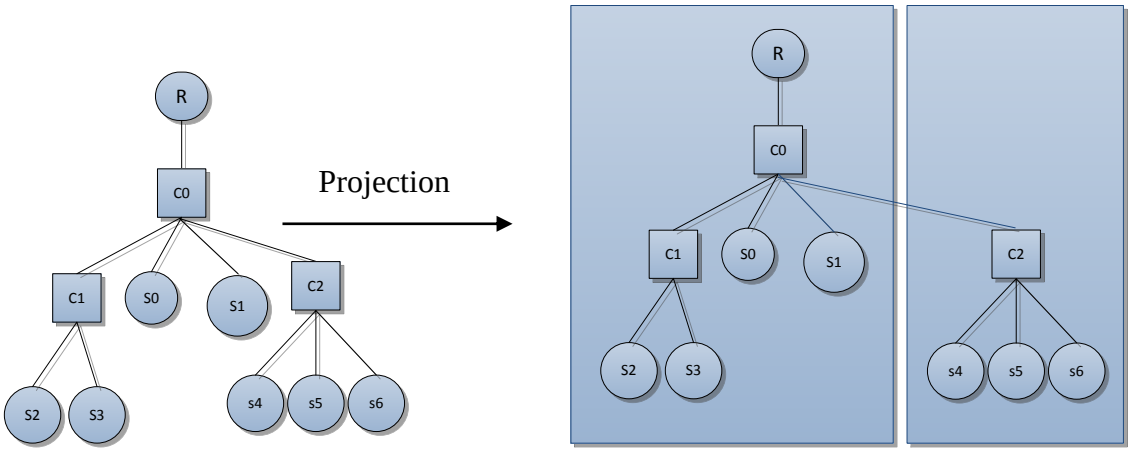


Figure 17. Projection synchrone

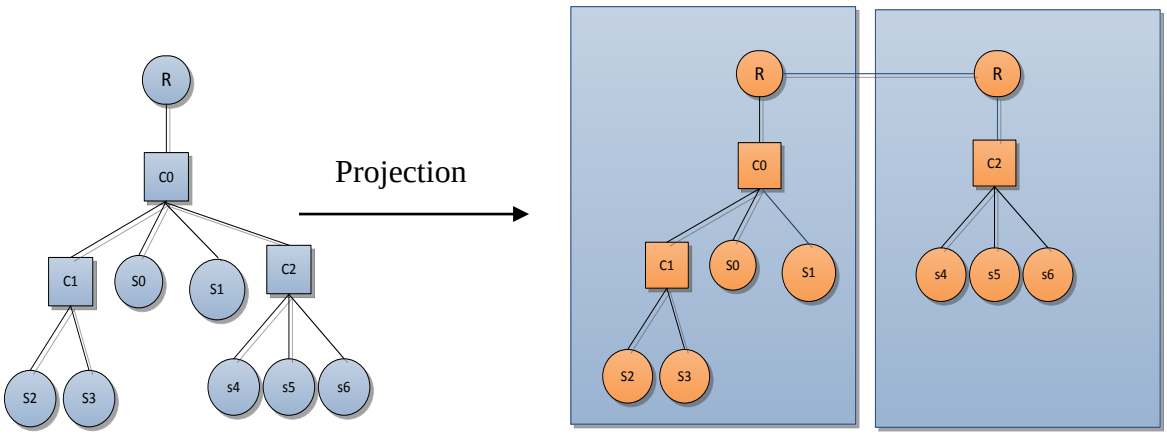


Figure 18. Projection asynchrone (modification des algorithmes)

Les projections synchrones adoptent, par essence, une démarche de SPD pessimiste. Elles ne modifient pas les algorithmes initiaux de simulation, sauf à introduire dans le code les mécanismes de communication inter processeurs. Toutefois, elles n'offre qu'un très faible potentiel de parallélisme, car toute la simulation est conduite par l'unique racine de l'arbre de simulation, et les seules exécutions concurrentes sont les acheminements de messages de

simulation le long des différentes branche de l'arbre. Ce type de parallélisation de DEVS n'offre pas beaucoup d'intérêt à nos yeux.

Les projections asynchrones, elles, correspondent à une parallélisation de DEVS à très fort potentiel de concurrence. Elles proposent des variantes pessimistes et optimistes dans la littérature. Toutes conduisent à une réingénierie en profondeur des implémentations existantes, soit par modification des algorithmes initiaux de simulation DEVS, soit par modification de la structure même de l'arbre de simulation (Figure 19) et par conséquent, par l'adoption de nouveaux algorithmes.

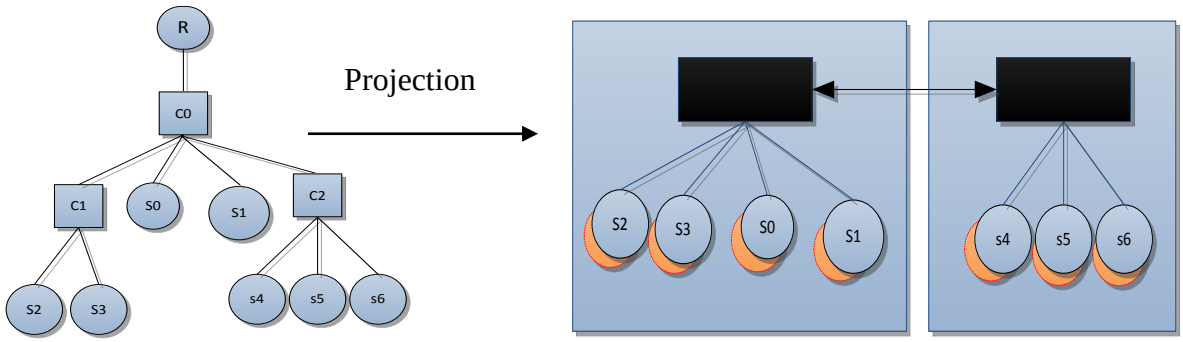


Figure 19. Projection avec modification du protocole de simulation

3.4.1. Projections synchrones

Les approches synchrones utilisent un gestionnaire central (coordinateur Racine) pour synchroniser la simulation entre les différents processeurs présents dans la simulation repartie. Seuls les évènements avec le même temps de simulation peuvent être parallélisés.

Un exemple d'algorithme synchrone est celui proposé dans James II [Himmelspach & Uhrmacher 2006]. Dans cet algorithme de simulation, des niveaux de hiérarchie virtuels sont ajoutés à l'arborescence. Un simulateur abstrait est défini comme solution générique, et deux options sont proposées : une avec des *threads* Java, et une autre sans thread. Pour le simulateur abstrait avec threads, certains coordinateurs sont substitués par un seul coordinateur virtuel et plusieurs sous-coordinateurs. Ceci permet de partitionner un grand modèle couplé en plusieurs modèles couplés virtuels. Cette approche combine la flexibilité du simulateur abstrait avec l'efficacité des simulateurs séquentiels en diminuant la quantité de threads nécessaires sur un processeur physique. Les composants intervenants dans la simulation sont : Root coordinator, Coordinator, Virtual Coordinator et Sub Coordinator. Il n'y a pas d'introduction de mécanismes de synchronisation de SPD, du fait que les branches ne se trouvant pas sur le processeur supportant le coordinateur Racine n'ont pas d'autonomie propre (pas de gestion locale d'une horloge).

Nous ne nous intéressons pas, dans notre effort de Recherche, à cette forme de projection.

3.4.2. Projections asynchrones

Parmi ceux qui utilisent le mécanisme de synchronisation asynchrones, la plus grande partie implémente l'approche optimiste. Les projections asynchrones de la littérature modifient, toutes, les algorithmes de tous les nœuds de base de la simulation (coordinateurs et

simulateurs) [Zeigler et al. 2000], [Seong et al. 1995]. La stratégie optimiste (sauvegarde des états, roll-back et envoi d'anti-messages) est implémentée sur chaque nœud. Ainsi, simulateur, coordinateur et racine sont modifiés pour donner respectivement simulateur optimiste, coordinateur optimiste et racine optimiste.

3.4.2.1. DEVS-Ada TW

DEVS-Ada/TW [Christensen & Zeigler 1990] (première tentative de combiner DEVS et le mécanisme Time Warp pour la simulation distribuée optimiste) est une approche asynchrone qui utilise le mécanisme Time Warp pour la synchronisation globale. Dans DEVS-Ada/TW le modèle hiérarchique DEVS ne peut être partitionné qu'au plus haut niveau de sa hiérarchie pour la simulation distribuée. Cela restreint la flexibilité du partitionnement des modèles.

3.4.2.2. DOHS

Le schéma DOHS (Distributed Optimistic Hierarchical Simulation) est une méthode DEVS répartie optimiste qui utilise le mécanisme de Time Warp. Il met en œuvre les types de nœud suivants, selon des algorithmes détaillés dans [Seong et al. 1995] :

- Un coordinateur de nœud, version parallèle du coordinateur racine, qui génère les messages pour les simulateurs abstraits du sous-arbre (parallel abstract simulator) se trouvant sur le processeur. Les algorithmes des automates de chaque sous arbre combinent les algorithmes du simulateur abstrait séquentiel et du Time Warp.
- Des *p-simulateurs* et des *p-coordinateurs*, versions parallèles des simulateurs et coordinateurs initiaux.
- Une file d'attente, nommée *DOHS-queue*, dans laquelle sont stockés les messages avant leur traitement.
- Un contrôleur de l'exécution de la simulation, appelé *DOHS-manager*.

3.4.2.3. DEVS Time Warp

Un autre simulateur réparti optimiste DEVS à base de Time Warp a été proposé dans [Zeigler et al. 2000]. Dans cette approche, le modèle global est partitionné de sorte que chaque partition corresponde à un modèle couplé existant. Chaque processeur dispose d'un coordinateur racine qui réalise le mécanisme de Time Warp. Il s'occupe aussi du calcul du GVT (Global Virtual Time) sur ce processeur et de la collecte des fossiles.

3.4.2.4. Risk-free DEVS

Cette approche proposée dans [Zeigler et al. 2000] est une optimisation du Time Warp, par réduction de la portée des Rollback, qui habituellement peuvent se solder par un effet boule de neige entrainant progressivement tous les nœuds dans un retour au passé. Bien que chaque nœud adopte un algorithme de type optimiste, les messages à expédier d'un processeur à un autre sont gardés jusqu'à ce qu'on puisse garantir qu'ils sont sains pour être traités par le processeur de réception.

3.4.2.5. PCD++

Il existe des solutions qui aplatissent l'arbre de simulation initial. Cela conduit à la réduction des coûts des échanges de messages, et par la suite à de meilleures performances [Wainer 2009]. C'est le cas du simulateur optimiste dans P-CD++ [Liu & Wainer 2007], une version optimiste de l'outil CD++ développé pour la simulation optimiste des modèles DEVS et Cell-DEVS. La mise à plat de l'arbre de simulation se fait au prix d'une profonde modification de l'algorithme de chaque composant.

3.4.2.6. EIT-EOT

Contrairement aux approches optimistes, peu de simulateurs DEVS répartis appartiennent à la classe conservative. Dans [Zeigler et al. 2000], une approche de simulation répartie conservative est décrite pour les modèles DEVS non-hiérarchiques. Sur chaque nœud on peut avoir plusieurs simulateurs DEVS, et à chacun d'eux est associé un nœud conservatif (qui contrôle le simulateur DEVS, et gère la distribution des messages aux autres nœuds). Chaque nœud conservatif maintient les estimations pour la date au plus tôt de la prochaine réception de message (EIT), et la date au plus tôt du prochain envoi de message (EOT). Le nœud ne doit jamais traiter une entrée avec une date inférieure à son EIT. Sur la base de cette estimation et de l'état local, il peut déterminer le lookahead qui sert à calculer la valeur EOT. Les valeurs EIT/EOT sont propagées entre les composants via les messages nuls. Lorsque la valeur EOT d'un composant change, il envoie des messages nuls à tous ses influencés pour leurs communiquer la nouvelle valeur. Les messages d'entrée en provenance des autres composants ne sont pas traités immédiatement, mais stockés dans une file d'événements, de sorte que l'ordre des dates de leur traitement est assuré.

Les coûts temporels de cette approche peuvent être importants en raison des messages nuls. Pour les réduire, plusieurs améliorations ont été proposées. Il en existe trois variantes : (1) envoi de messages nuls de temps en temps, (2) envoi de messages nuls uniquement en cas de blocage, et (3) envoi de messages nuls uniquement sur demande. Chacune des approches a ses avantages et ses inconvénients. Par ailleurs, les performances d'une approche conservative dépendent toujours d'un bon lookahead.

3.4.2.7. CCD++

Pour surmonter certaines limites du simulateur DEVS pessimiste proposé dans [Zeigler et al. 2000], à savoir le nombre élevé de calcul de EIT et EOT et par conséquent le nombre élevé de messages nuls échangés entre les composants, un algorithme conservatif basé sur le mécanisme de synchronisation classique de Chandy, Misra et Bryant a été proposé dans [Jafer & Wainer 2010]. Dans cette approche, le mécanisme de synchronisation pessimiste est implémenté au plus haut niveau de la hiérarchie sur chaque processeur et les calculs des EIT/EOT sont remplacés par le seul calcul de lookahead, réduisant ainsi considérablement le nombre de messages nuls générés. Le simulateur abstrait implémenté dans CCD++ (version conservative de CD++ pour la simulation des modèles DEVS et Cell-DEVS) est basé sur une version révisé du simulateur abstrait P-DEVS proposé dans [Liu & Wainer 2007].

3.4.3. Problématique de partitionnement

Le partitionnement de modèle est le processus d'agrégation ou de décomposition de modèles en un ensemble de blocs. Comment définir la meilleure partition est l'une des questions les plus importantes en SPD car il affecte directement la performance de la simulation. Trouver la partition optimale est un problème NP-complet.

Le Partitionnement Générique de Modèles (GMP, pour Generic Model Partitioning) est un algorithme de partitionnement des modèles DEVS hiérarchiques [Park 2003]. Il décompose un modèle hiérarchique donné en plusieurs blocs en se basant sur le concept de coût de partitionnement. L'algorithme comporte deux parties : (1) l'analyse des coûts et la construction de l'arbre de coût, et (2) la partition de l'arbre des coûts. Un coût peut être de plusieurs formes en fonction du système :

- Si le système est dédié aux opérations d'entrées/sorties, alors le coût du système est le nombre d'interfaces d'entrées/sorties ($|X|*|Y|$).
- Si la performance du système est liée à sa complexité, alors le coût du système est représenté par le nombre d'états internes ($|S|$).
- Si la performance est liée à la fois à la complexité et aux opérations d'entrées/sorties, alors le coût peut être le produit du nombre d'interfaces d'entrées/sorties par le nombre d'états internes du modèles ($|X|*|Y|*|S|$).
- Si la performance est liée aux comportements dynamiques du système, alors le coût est représenté par le nombre de transitions internes $(|\delta_{int}|)$.

Dans [Kim et al. 1998] les auteurs ont proposé un algorithme de répartition pour la simulation repartie optimiste des modèles modulaires et hiérarchiques DEVS. Cet algorithme appelé HIPART (pour Hierarchical partitioning) vise les objectifs suivants : (1) équilibrer des charges de calcul, (2) maximiser l'exécution parallèle des modèles indépendants, et (3) minimiser la communication inter processeurs. Pour maximiser l'exécution parallèle des modèles indépendants, l'algorithme proposé utilise les informations de la structure hiérarchique des modèles disponibles dans la méthodologie de conception hiérarchique. Ainsi, la stratégie de base de l'algorithme proposé est d'insérer le maximum de composant dans le même bloc si possible (i.e., partitionner l'arbre de composition hiérarchique au niveau le plus haut possible). Puisque l'algorithme proposé supporte le schéma DOHS présenté précédemment, les auteurs ont fait l'hypothèse suivante pour un algorithme de simulation : pour un nœud qui n'est pas une feuille (c'est-à-dire un coordinateur), au moins un descendant (simulateur ou coordinateur) doit être dans le même bloc. En d'autres termes, on ne peut pas avoir un coordinateur sur un nœud sans simulateur fils.

Nous n'adressons pas, dans ce travail, la recherche de la partition optimale. Notre travail vient en aval du choix de partitionnement fait, et s'inscrit dans l'effort de réalisation de cette partition de son expression conceptuelle à son implémentation sur une architecture parallèle.

3.4.4. Supports au protocole DEVS réparti

Rappelons l'importance, lorsqu'on en vient à l'implémentation d'une solution de SPD, des mécanismes de communication inter processeurs pour supporter le protocole de transmission des messages entre nœuds de simulation (section 2.5), ces derniers ayant vocation à s'exécuter sur des processeurs différents, géographiquement distribués ou non.

Citons, dans le cas de DEVS réparti, les travaux suivants en la matière : DEVS/CORBA [Zeigler et al. 1999 a], DEVS/HLA [Zeigler et al. 1999 b], DEVS/Grid [Seo et al 2004], DEVSCluster [Kim & Kang 2004], DEVS/P2P [Cheon et al. 2004] et DEVS/RMI [Zhang et al. 2006].

Dans le cadre de cette thèse, nous avons implémenté deux solutions : l'une à base de CORBA en Java (RMI-CORBA) et l'autre à base de Machine Virtuelle MPI (une forme à l'intersection du P2P et du Cluster/Grid).

4. Ingénierie Dirigée par les Modèles

L'ingénierie dirigée par les modèles (IDM, ou Model Driven Engineering -MDE) [Combemale 2008], est une approche d'ingénierie dans laquelle tout ou partie d'un système est engendrée à partir de différents modèles, liés les uns aux autres, et exprimés au moyen d'un ou de plusieurs langages de modélisation différents (appelés DSLs pour Domain Specific Languages) [Gray et al. 2007], que les aspects chronologiques ou technologiques du développement nécessitent. Pour définir ces DSLs, il est fait appel à la méta-modélisation, et pour relier les différents modèles il est fait appel à la transformation de modèle. C'est sur ces principes que se base l'organisation de l'approche de modélisation de l'OMG. Elle est décrite sous une forme pyramidale à 4 niveaux, comme le montre la Figure 20.

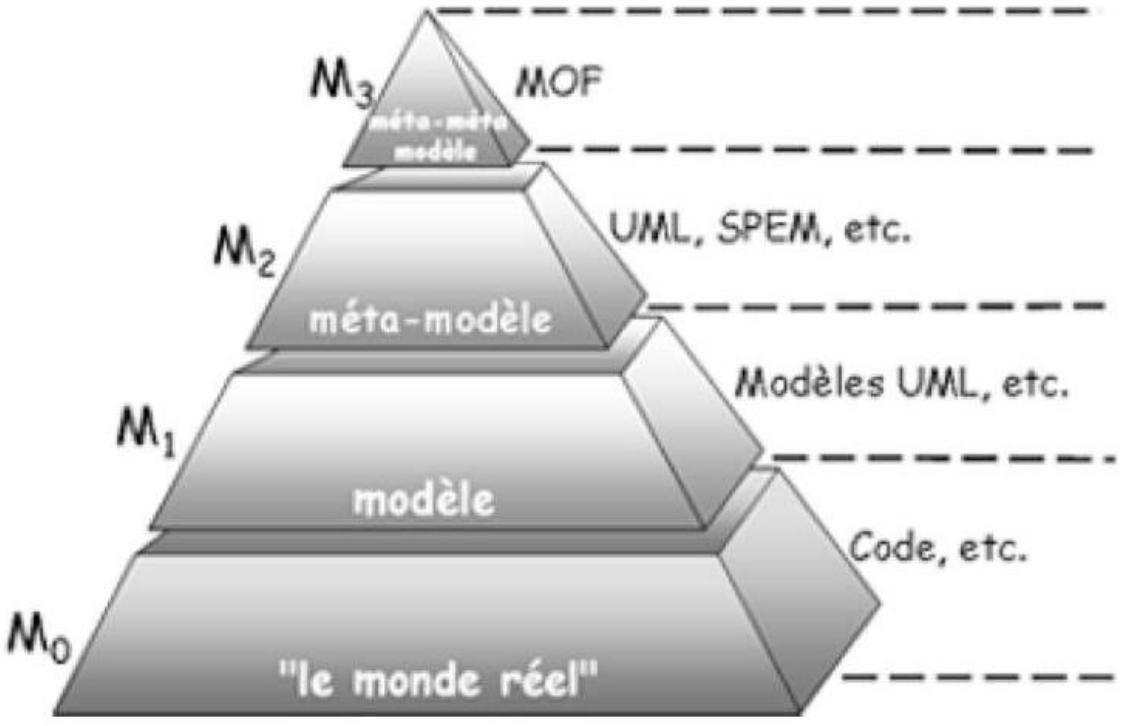


Figure 20. Pyramide de modélisation de l'OMG

Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. L'OMG préconise UML comme langage de modélisation, et les DSLs additionnels utilisés sont souvent des profils UML (i.e., des spécialisations d'UML à un domaine précis). Les méta-modèles permettant la définition de ces modèles constituent le niveau M2 (UML et ses DSLs dérivés). Enfin, le méta-méta-

modèle, unique et méta-circulaire, est représenté au sommet de la pyramide (niveau M3). Un méta-méta-modèle est un modèle qui décrit un langage de méta-modélisation, i.e., les éléments nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même. L'OMG préconise le MOF (Meta-Object Facility) [OMG 2006].

4.1. Architecture MDA

Une des raisons majeures de l'apparition des architectures dirigées par les modèles repose sur la volonté de décrire le problème et sa solution par des CIM (Computational Independent Models) et le savoir-faire ou la connaissance métier dans des modèles abstraits indépendants des plates-formes (PIM - Platform Independent Models). Ayant isolé le savoir-faire métier dans des PIM, on a besoin soit de transformer ces modèles en d'autres PIM (besoin de raffinement), soit de produire ou de créer des modèles PSM (Platform Specific Models) ciblant une plate-forme d'exécution spécifique. CIM, PIM et PSM se situent au niveau M1 de la pyramide. Les DSLs, eux se trouvent au niveau M2, ainsi que les définitions de transformation entre DSLs. Un exemple d'une telle architecture, et probablement la plus emblématique est MDA [Kleppe et al. 2003].

Le MDA se découpe en quatre couches de standards utilisés, comme l'indique la Figure 21. Au centre, se trouvent les standards UML, MOF et CWM (pour Common Warehouse Metamodel). Dans la couche suivante, se trouve le standard XMI (pour XML Metadata Interchange) qui permet le dialogue entre les middlewares (Java, CORBA, .NET et web services). La troisième couche contient les services qui permettent de gérer les évènements, la sécurité, les répertoires et les transactions. Enfin, la dernière couche propose des frameworks spécifiques au domaine d'application (Finance, Télécommunication, Transport, Espace, médecine, commerce électronique, manufacture, …). Ainsi, en partant de la couche centrale, un architecte logiciel dirigera son application en évoluant de couche en couche pour aller vers le domaine d'application qui l'intéresse.

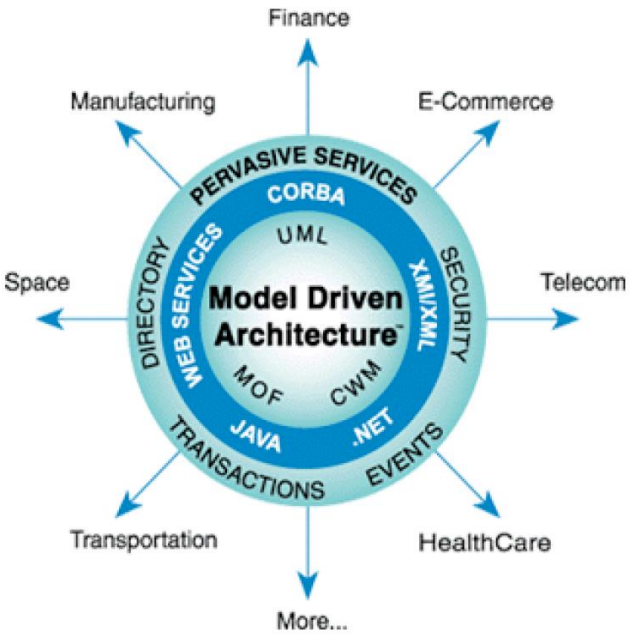


Figure 21. Architecture du MDA

Plus précisément, le MDA préconise l'élaboration de 3 types de modèles inter reliés :

- Un modèle d'exigence (appelé CIM pour Computation Independent Model) dans lequel aucune considération informatique n'apparait.
- Un ou plusieurs modèles d'analyse et de conception (appelés PIMs pour Platform Independent Models), qui sont indépendants des détails techniques des plateformes d'exécution.
- Un ou plusieurs modèles pour la génération de code (appelés PSMs pour Platform Specific Models).
- Un ou plusieurs modèles de description de plateforme hôte (appelés PDMs pour Platform Description Model).

Le passage de modèle à modèle fait intervenir des mécanismes de transformation de modèle et, dans certains cas, un modèle de description de la plateforme (appelé PDM pour Platform Description Model). Cette démarche s'organise donc selon un cycle de développement « en Y» que montre la Figure 22.

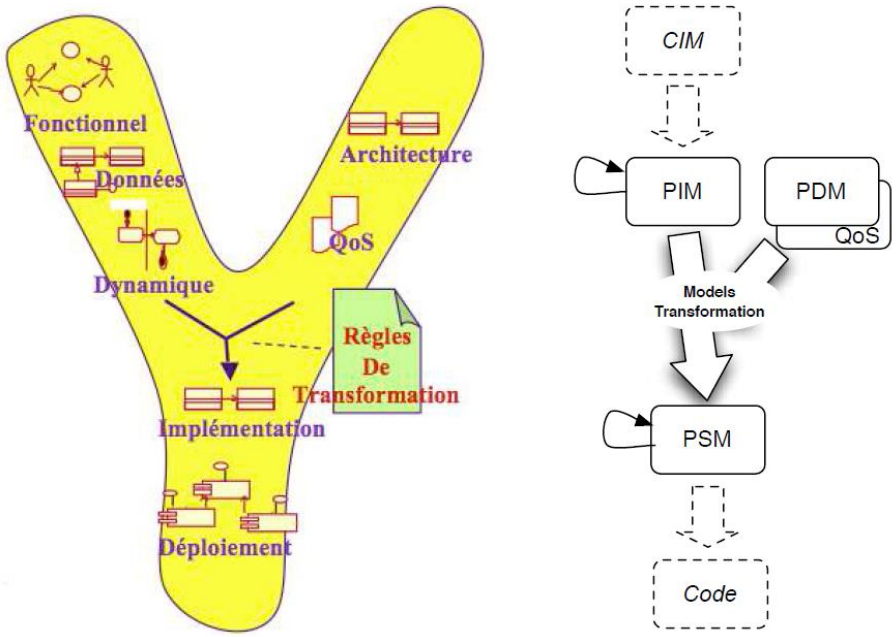


Figure 22. MDA : Un processus en Y dirigé par les modèles

4.1.1. CIM (Computation Independent Model)

C'est le modèle métier (ou modèle du domaine). Le CIM permet de représenter ce que le système devra faire dans son environnement, mais sans rentrer dans le détail de sa structure opérationnelle, ni de son implémentation. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps et il est modifié uniquement si les connaissances ou les besoins métier changent. Le savoir-faire est donc recentré sur le CIM au lieu de la technologie d'implémentation.

4.1.2. PIM (Platform Independent Model)

Logique métier (ou modèle de conception), le PIM permet de représenter tout ou une partie de la structurelle opérationnelle d'un système, de manière indépendante de toute technologie de déploiement (EJB, CORBA, .NET,…). Il représente le fonctionnement des entités et des

services. Plusieurs niveaux de PIM peuvent être définis, les uns raffinant les autres. Le PIM peut contenir des informations sur la persistance, les transactions, la sécurité,… Ces concepts permettent de transformer plus précisément le modèle PIM vers le modèle PSM.

4.1.3. PSM (Platform Specific Model)

C'est le modèle de l'implémentation. Le PSM décrit comment le système utilisera la plate-forme technologique dont il dépend et sert de base à la génération du code exécutable. Plusieurs niveaux de PSM peuvent être décrits, les uns affinant les autres jusqu'à l'obtention du code dans un langage spécifique (Java, C++, C#, etc.).

4.1.4. PDM (Platform Description Model)

Il contient des informations pour la transformation de modèles vers une plate-forme en particulier et il est spécifique de celle-ci. C'est un modèle de transformation qui va permettre le passage du PIM vers le PSM.

4.1.5. Du CIM au PSM

Le passage entre CIM, PIM et PSM se fait par une suite de transformations, automatisées au moyen d'outils logiciels, ou semi-automatisées voire simplement assistées. La Figure 23 montre un exemple de réalisation de plusieurs applications à partir d'un même CIM.

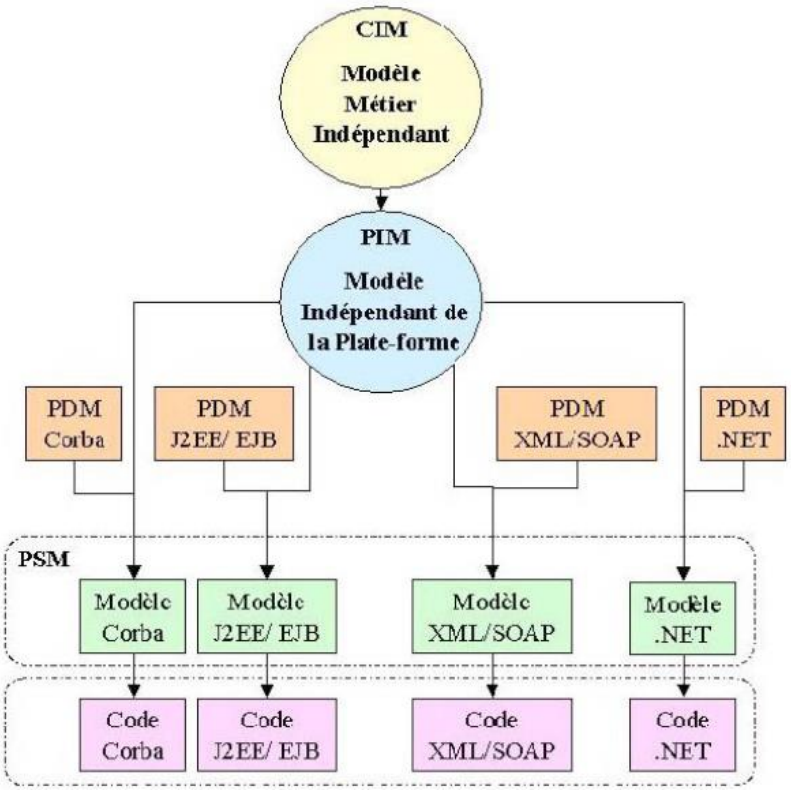


Figure 23. Exemples de passage de CIM à PSM

4.2. Transformation de modèles

Une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources [Bézivin 2004]. Dans [Blanc 2005] trois approches de transformations de modèles sont retenues :

- l'approche par programmation, qui consiste à utiliser les langages de programmation (les langages orientés objet en particulier) pour décrire la transformation ;
- l'approche par template, qui consiste à définir des canevas (ou templates) des modèles cibles souhaités, i.e., des modèles paramétrés tels que l'exécution d'une transformation consiste à prendre un modèle template et à remplacer ses paramètres par les valeurs d'un modèle source ;
- l'approche par modélisation, qui consiste à modéliser les transformations de modèles, en exprimant leur indépendance vis-à-vis des plates-formes d'exécution. Le standard QVT de l'OMG a été élaboré dans ce cadre et a pour but de définir un méta-modèle permettant l'élaboration des modèles de transformation de modèles. A titre d'exemple, cette approche est celle promue par le langage de transformation de modèles ATL.

Dans l'approche par modélisation, que nous privilégions pour cette thèse, la transformation se fait par l'intermédiaire de règles de transformations qui décrivent la correspondance entre les entités du modèle source et celles du modèle cible. En réalité, la transformation se situe entre les méta-modèles source et cible qui décrivent la structure des modèles cible et source. Le moteur de transformation de modèles prend en entrée un ou plusieurs modèles sources, applique la fonction de correspondance définie entre les concepts des modèles source et cible au niveau de leurs méta-modèles, et crée en sortie un ou plusieurs modèles cibles. La figure 24 illustre cette démarche [Farail et al. 2006].

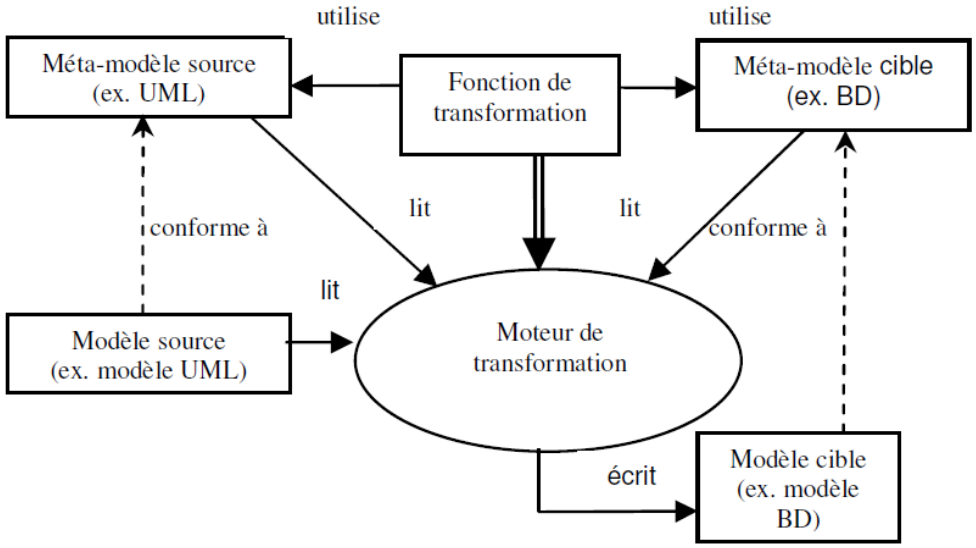


Figure 24. Schéma de base d'une transformation de modèles

4.2.1. Types de transformation

Les transformations opérées sur un modèle source produisent un modèle cible dont le niveau de finesse (ou de détails) est plus ou moins élevé que celui de la source.

On distingue les types de transformation suivants [Csarnecki & Helsen 2003] :

- La transformation 1 vers 1, qui associe à tout élément du modèle source un élément du modèle cible. Un exemple typique de cette situation est la transformation d'une classe UML munie de ses opérations et de ses attributs en une classe homonyme en Java.
- La transformation M vers N, qui prend en entrée un ensemble d'éléments du modèle source et produit un ensemble d'éléments du modèle cible. Les décompositions de modèles (1 vers N) et fusions de modèles (N vers 1) en sont des cas particuliers.
- La transformation de mise à jour (ou transformation sur place), qui consiste à modifier un modèle par ajout, modification ou suppression d'une partie de ses éléments. Dans ce type de transformation, le modèle source est aussi la cible. Un exemple est la restructuration de modèles (ou *Model Refactoring*) qui consiste à réorganiser les éléments du modèle source afin d'en améliorer sa structure ou sa lisibilité.

4.2.2. Outils de transformation

La littérature propose 4 sortes d'outils de transformation de modèles [Garredu et al. 2014] :

- Les outils génériques, où se retrouvent les outils de la famille XML (XLST, Xquery, etc. [W3C 2015]) et les outils de transformations de graphes qui sont principalement utilisés dans le monde académique [Ehrig et al. 2005], [Atkinson & Kuhne 2003]. Ces outils, bien que populaires, s'avèrent peu adaptés à des modèles de grande complexité.
- Les outils intégrés aux AGL (Ateliers de Génie Logiciel), où se retrouvent les outils commerciaux (MDA Modeler, IBM Rational Software Modeler, etc.) [Blanc 2005] et certains outils du monde académique (FUJABA [Burmester et al. 2004]). Ces outils ont eux aussi un bon niveau de maturité, mais présentent eux aussi certaines limitations avec les modèles complexes.
- Les langages/outils dédiés à la transformation de modèles, où se retrouvent des langages conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standard (comme ATL [Jouault & Kurtey 2005]).
- Les outils de méta-modélisation, dans lesquels la transformation de modèles revient à l'exécution d'un méta-programme (Kermeta [Muller et al. 2005], MetaEdit+ [Tolvanen & Rossi 2003], Eclipse EMF/Ecore [Budinsky et al. 2003], TOPCASED [Farail et al. 2006]).

Certains outils sont à l'intersection de ces catégories. Le plus emblématique est probablement ATOM3 [Delara & Vangheluwe 2002].

5. Conclusion

Nous avons, dans ce chapitre, présenté les concepts et techniques qui sous-tendent notre travail.

Tout d'abord, nous nous sommes intéressés aux approches de simulation répartie (dite aussi simulation parallèle et distribuée, ou SPD). La maturité théorique de ce domaine se traduit par l'existence de plusieurs algorithmes, les uns dits conservatifs ou pessimistes, les autres dits optimistes, toutes mettant l'accent sur des échanges de message pour synchroniser les différents processeurs impliqués dans l'exécution du programme de simulation.

Nous avons ensuite présenté le formalisme DEVS, qui s'est imposé comme le langage universel de spécification des systèmes à événements discrets, et qui est au cœur de nos travaux. Les algorithmes originels de DEVS sont destinés à une exécution séquentielle, mais le besoin de plus en plus pressant d'optimiser les performances d'exécution face à des modèles à taille et à complexité croissante, ont conduit à l'élaboration, par plusieurs groupes de Recherche, de solutions parallèles. Toutefois, ces dernières passent toutes par des altérations profondes des algorithmes du protocole de simulation. La conséquence directe en est qu'il n'est pas possible de réutiliser des implémentations existantes de DEVS et de leur adjoindre les mécanismes de synchronisation nécessaires à une simulation répartie, sans produire des efforts importants de réingénierie logicielle.

Nous avons enfin présenté les concepts principaux de l'Ingénierie Dirigée par les Modèles (IDM). Cette approche de Génie Logiciel avancé préconise de décrire au travers de modèles, concepts, et langages, à la fois le problème posé et sa solution. Un de ses intérêts est de permettre d'automatiser la transformation de modèle, de son expression conceptuelle à son code. Nous nous inspirons de l'architecture MDA pour proposer un cadre de parallélisation de simulateurs DEVS existants, guidé par les principes de méta-modélisation et de transformation de modèles que prône cette architecture d'IDM.

Le chapitre suivant présente la méthode que nous proposons pour paralléliser un simulateur DEVS séquentiel existant, par adjonction de composants de synchronisation, et sans modification des composants existants. L'intérêt d'une telle méthode est de réduire considérablement les coûts de développement et la complexité du passage d'une implémentation séquentielle DEVS à sa contrepartie répartie. Plusieurs plateformes existantes sont concernées. Dans le chapitre d'après, nous envisageons de tirer profit des possibilités de l'Ingénierie Dirigée par les Modèles pour systématiser la démarche de parallélisation des simulateurs séquentiels DEVS basée sur notre approche d'injection de composants de synchronisation sans altération du code existant. Cette méthodologie sera éprouvée sur notre propre package de simulation DEVS appelé SimStudio, et nous ferons une étude de performances des solutions construites en utilisant deux cas d'étude comme banc d'essai. La mise en œuvre et l'étude de performances sont présentées dans les chapitres ultérieurs.

Chapitre III.
APPROCHE PAR MANTEAUX

1. Introduction

Comme nous l'avons déjà vu au chapitre précédent, DEVS sépare le modèle de son simulateur. Le modèle provient de l'activité de modélisation du système réel, et le simulateur est un arbre construit à partir de la hiérarchie du modèle. Cet arbre produit la sémantique opérationnelle de DEVS dans un contexte d'exécution séquentielle. Pour une exécution répartie, l'arbre initial de simulation est remplacé par un graphe de simulation dans lequel plusieurs arbres de simulation séquentielle DEVS sont fédérés. La figure 25 résume les éléments de base du contexte d'exécution d'un modèle DEVS.

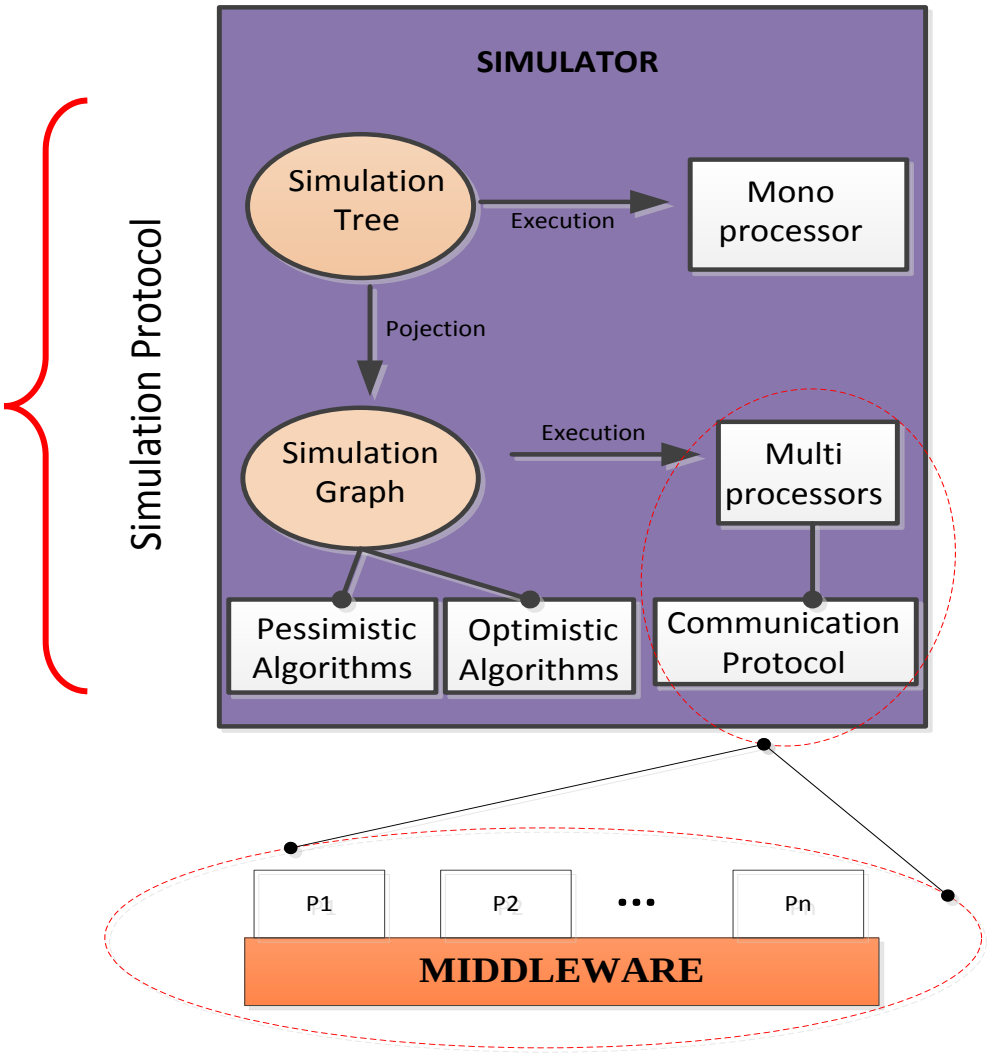


Figure 25. Eléments de simulation répartie avec DEVS

Pour une exécution séquentielle, il existe de nos jours plusieurs implémentations du formalisme DEVS. Les modèles de grande taille, eux, nécessitent le recours à la simulation répartie, mais la répartition du schéma de simulation DEVS n'est pas triviale. L'une des difficultés est la réutilisation des simulateurs séquentiels existants. La littérature montre que les techniques adoptées passent toutes par une profonde modification des algorithmes liés au protocole initial de simulation DEVS (y compris, dans certains cas, une modification de la hiérarchie initiale entre simulateurs et coordinateurs), et se traduisent par des efforts d'implémentation non négligeables.

L'objectif de ce travail est de proposer, dans un premier temps, une démarche permettant de mettre en œuvre les approches de SPD par-dessus une implémentation séquentielle DEVS existante, sans modifier le schéma de simulation déjà en place (donc sans altérer le code en profondeur). Dans cette approche, la fédération des arbres de simulation est réalisée grâce à une couche horizontale de SPD que nous appelons manteaux.

Une fois le schéma de simulation correspondant à cette configuration bien défini dans ce chapitre, nous nous intéresserons dans un deuxième temps (chapitre suivant) à la transformation (que nous appelons projection) systématique et progressive de l'arbre de simulation en graphe de simulation [Adegoke et al. 2011], [Adegoke et al. 2013].

2. Graphe de simulation

Le graphe de simulation est la sémantique opérationnelle de DEVS dans un contexte réparti, de la même manière que l'arbre de simulation est sa sémantique opérationnelle dans un contexte séquentiel.

Prenons l'exemple du modèle DEVS donné par la Figure 26 (montrant uniquement son architecture, sans les couplages). Sa sémantique opérationnelle dans un contexte séquentiel est donnée par la Figure 27. Une hiérarchie de base est d'abord obtenue en en faisant correspondre à chaque composant du modèle un nœud de l'arbre (un simulateur pour chaque modèle atomique et un coordinateur pour chaque modèle couplé). Puis une racine est placée au sommet de cette hiérarchie pour gérer le temps global simulé.

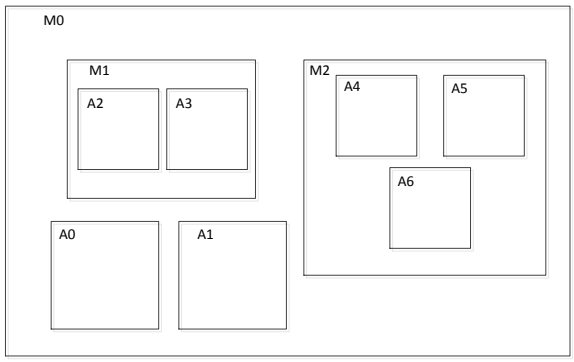


Figure 26. Exemple d'architecture de modèle DEVS

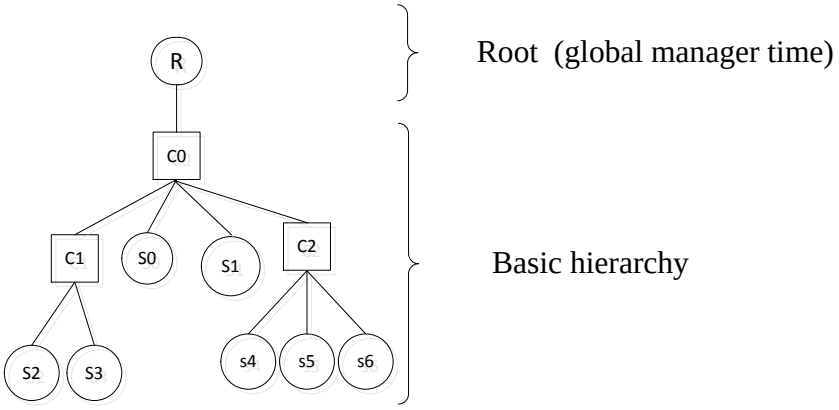


Figure 27. Déploiement séquentiel de DEVS

Dans notre approche répartie, nous créons un graphe composé d'arbres, chacun assigné à un processeur différent. La Figure 28 montre le schéma réparti dans le cas de 2 processeurs. La répartition est réalisée par une couche supplémentaire (matérialisée par un réseau de manteaux) qui garantit la modularité de l'approche et la réutilisabilité de la hiérarchie de base dans différentes stratégies d'exécution. Chaque arbre possède un manteau et une racine. La racine est le gestionnaire du temps comme dans le cas séquentiel et le manteau est chargé de la synchronisation. Les communications inter-arbres se font donc uniquement à travers les manteaux. L'un des principes directeurs étant de ne pas affecter les algorithmes de simulation déjà définis dans le cas séquentiel, les relations de la hiérarchie de base sont conservées pour les nœuds se trouvant dans le même arbre, et les messages qui circulent entre eux dans le protocole initial restent inchangés.

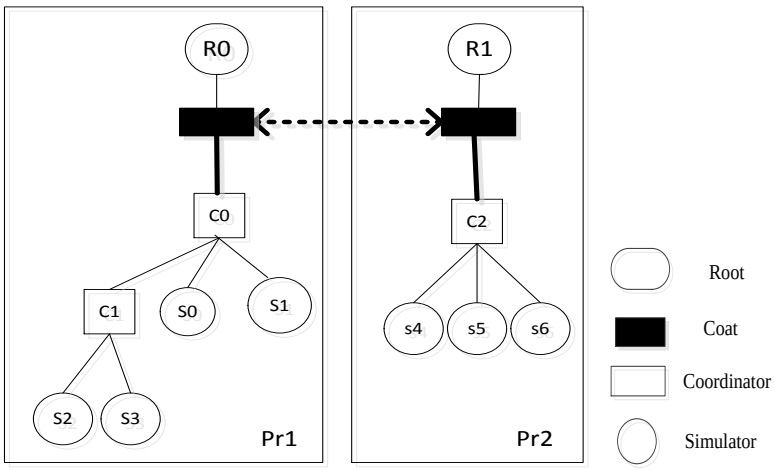


Figure 28. Déploiement réparti de DEVS sur deux processeurs

3. Composant Manteau

Nous allons définir ici quelques concepts utilisés partout dans notre travail.

3.1. Relations initiales dans l'arbre de simulation

En utilisant l'arbre de simulation de la Figure 29 comme exemple illustratif, nous convenons que :

Définition 1 : Le père (ou parent) d'un nœud n1 est le nœud n2 se trouvant dans la même branche et à un niveau au-dessus de n1 dans la hiérarchie (la racine de l'arbre définit le niveau le plus haut). Par exemple, *Parent(C1) = Parent(S0) = Parent(C2) = C0.*

Définition 2 : Le nœud n2 est un fils du nœud n1 si et seulement si n1 est le père de n2. Par exemple, *Children(C1) = {C3, S1, S2}.*

Définition 3 : Les ancêtres (ou grands-parents) d'un nœud sont les nœuds se trouvant dans la même branche et à au moins deux niveaux au-dessus de ce nœud dans la hiérarchie, à l'exception de la racine. Par exemple, *GrandParent(S4) = {C1, C0}.*

Définition 4 : Le nœud n2 est un descendant (ou GrandChildren) du nœud n1 si et seulement si n1 est un ancêtre de n2. Par exemple, *GrandChildren(C3) = {S4, C6, S5, S14, S15, S16}.*

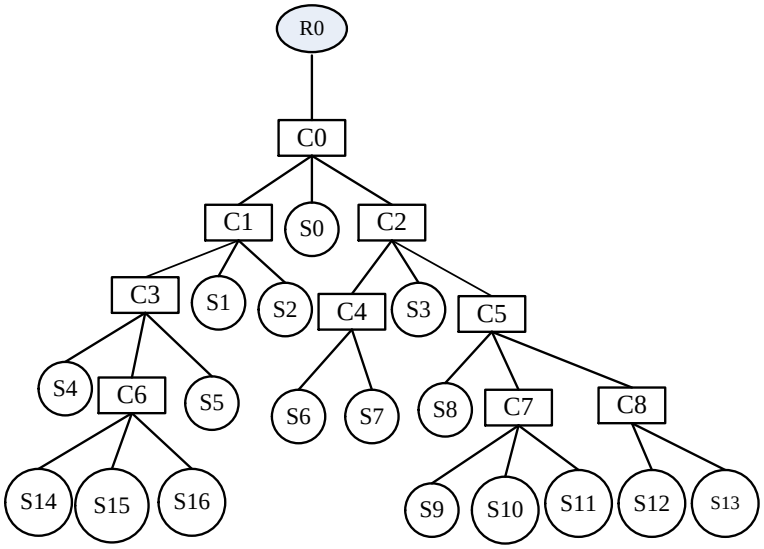


Figure 29. Arbre de simulation

3.2. Relations nouvelles dans le graphe de simulation

Les relations père/fils, entre les nœuds dans la hiérarchie de base, sont conservées après la répartition de l'arbre en graphe. Toutefois, si le fils n2 d'un nœud n1 devient distant du fait de la répartition, alors c'est le manteau contrôlant l'arbre dans lequel se trouve n1 qui fait office de fils par procuration pour n1, pour gérer les envois de message de n1 vers n2. Ce manteau transmet alors les messages à envoyer au manteau qui contrôle l'arbre de n2, qui à son tour les délivre à n2. Afin de bien capturer ces aspects et d'autres aspects relatifs, nous introduisons ici une série de relations nouvelles.

Définition 5 : Le parent réel d'un manteau est la racine de son arbre de simulation. Par exemple, dans la Figure 30, le manteau Ct0 contrôle l'arbre où se trouvent C1 et C2, alors que le fils distant C3 de C1 se trouve sur un arbre contrôlé par le manteau Ct1, et que le fils distant C5 de C2 se trouve sur un arbre contrôlé par le manteau Ct2. Nous avons donc :
RealParent(Ct0) = R0
RealParent(Ct1) = R1
RealParent(Ct2) = R2

Définition 6 : Le parent virtuel d'un manteau est le manteau du parent ou d'un grand parent, dans la hiérarchie de base, du coordinateur qu'il contrôle. Par exemple, dans la Figure 31, pour les mêmes raisons que précédemment, nous avons :
VirtualParent(Ct1) = VirtualParent(Ct2) = Ct0

Définition 7 : Le fils réel d'un manteau est le nœud (coordinateur ou simulateur) qu'il contrôle directement. Par exemple, dans la Figure 32, nous avons :
RealChild(Ct1) = C5
RealChild(Ct2) = C3
RealChild(Ct0) = C0

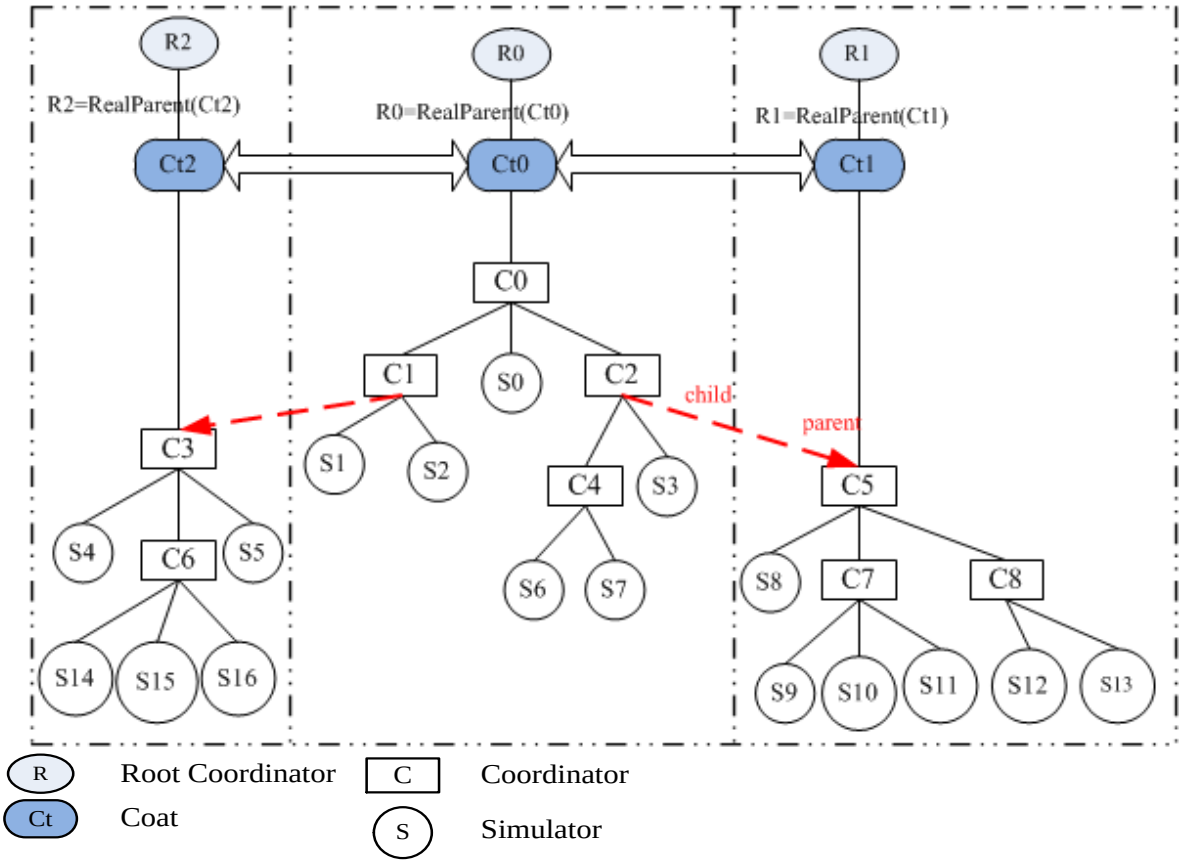


Figure 30. Parent réel d'un manteau

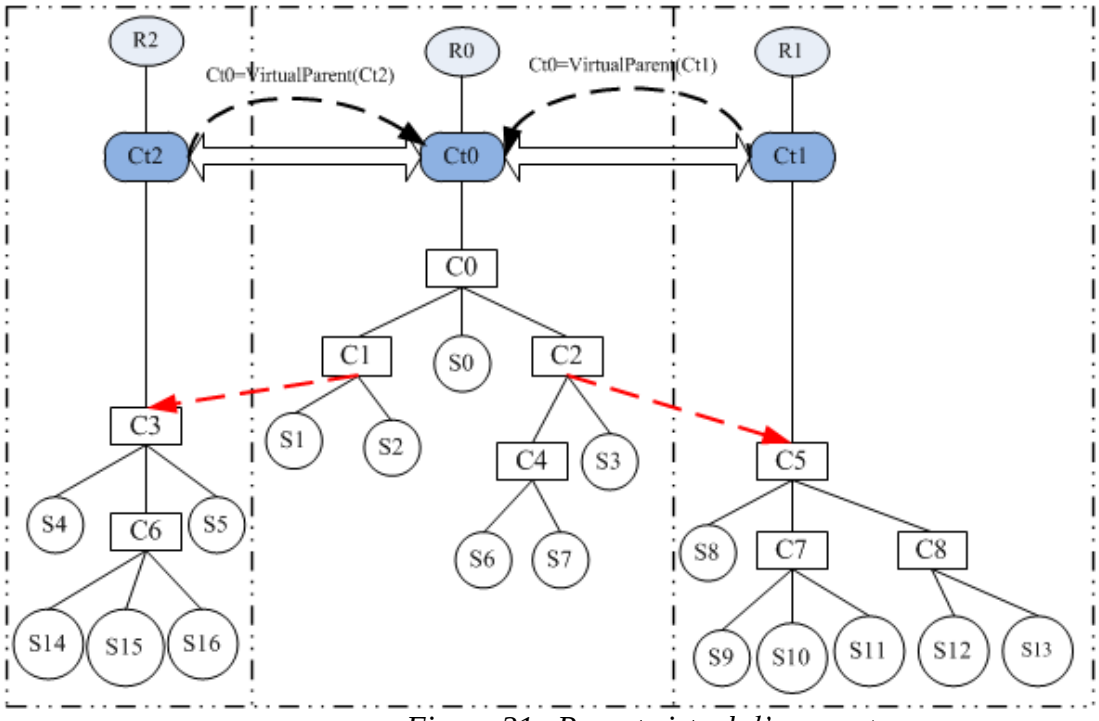


Figure 31. Parent virtuel d'un manteau

Définition 8 : Un fils virtuel d'un manteau est tout manteau qui contrôle l'arbre où se trouve un fils distant ou un descendant distant de son fils réel dans la hiérarchie de base. Par exemple, dans la Figure 33, nous avons : *VirtualChild(Ct0) = {Ct1, Ct2}*

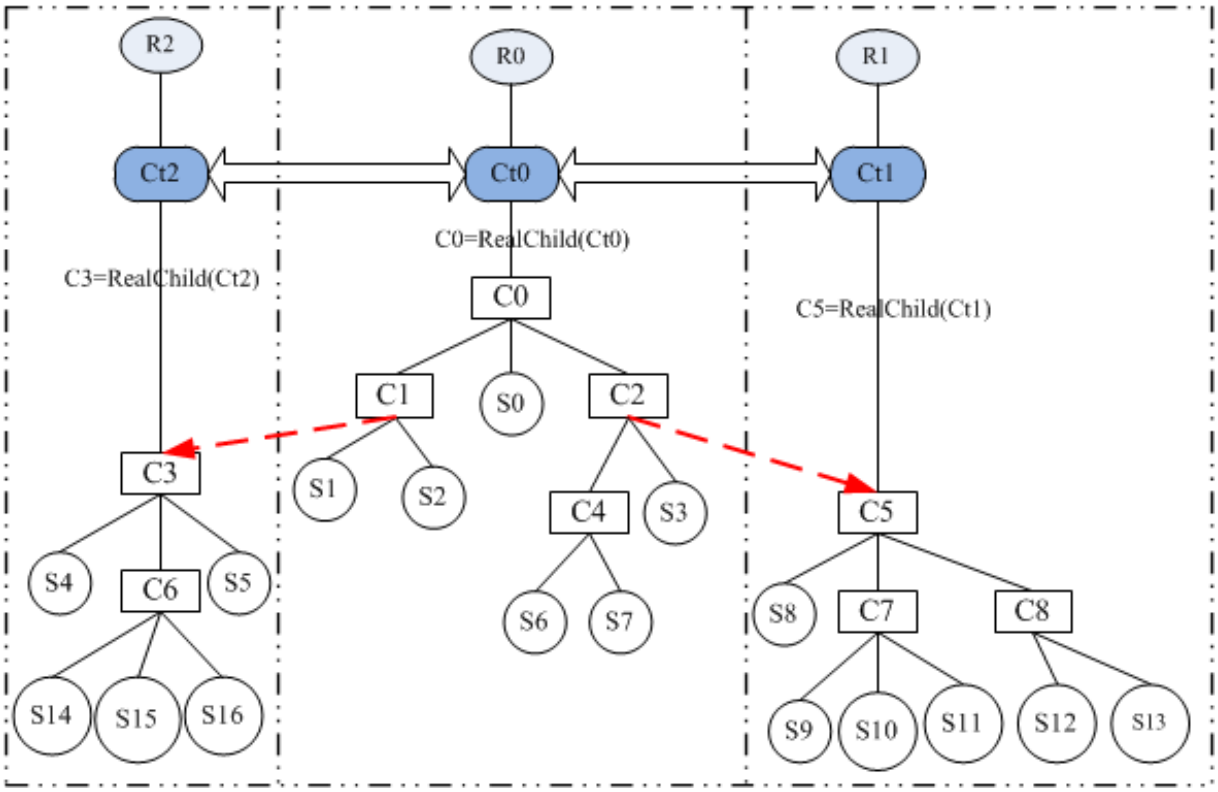


Figure 32. Fils réel d'un manteau

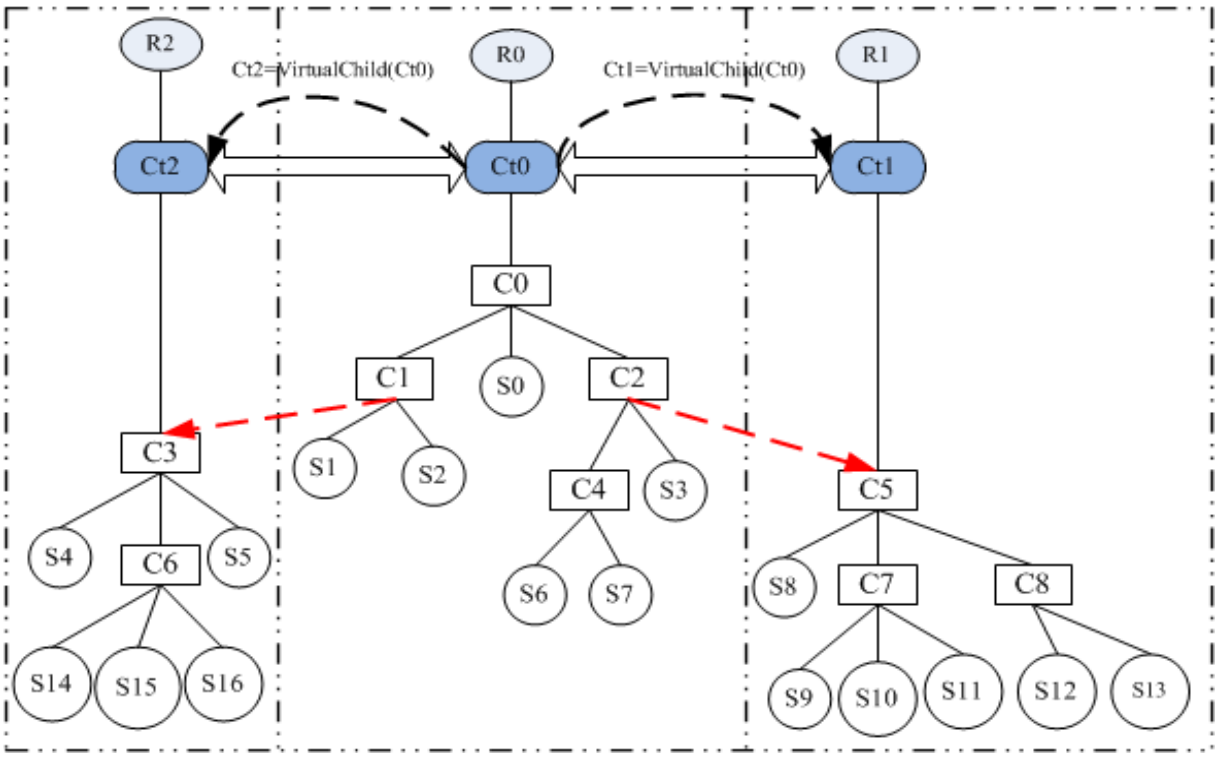


Figure 33. Fils virtuel d'un manteau

Définition 9 : Les parents procurateurs d'un manteau sont les nœuds (fils réel du manteau ou descendants de ce fils réel) coordinateurs qui ont au moins un fils distant. Ainsi, dans chaque arbre, les fils et parent distants d'un nœud sont remplacés par le manteau de cet arbre qui devient respectivement fils et parent par procuration de ce nœud. Par exemple, dans la Figure 34, nous avons :

ProcuratorParent(Ct0) = {C1, C2}
Children(C1) = {S1, S2, Ct0}
Children(C2) = {C4, S3, Ct0}
Children(C0) = {C1, S0, C2}

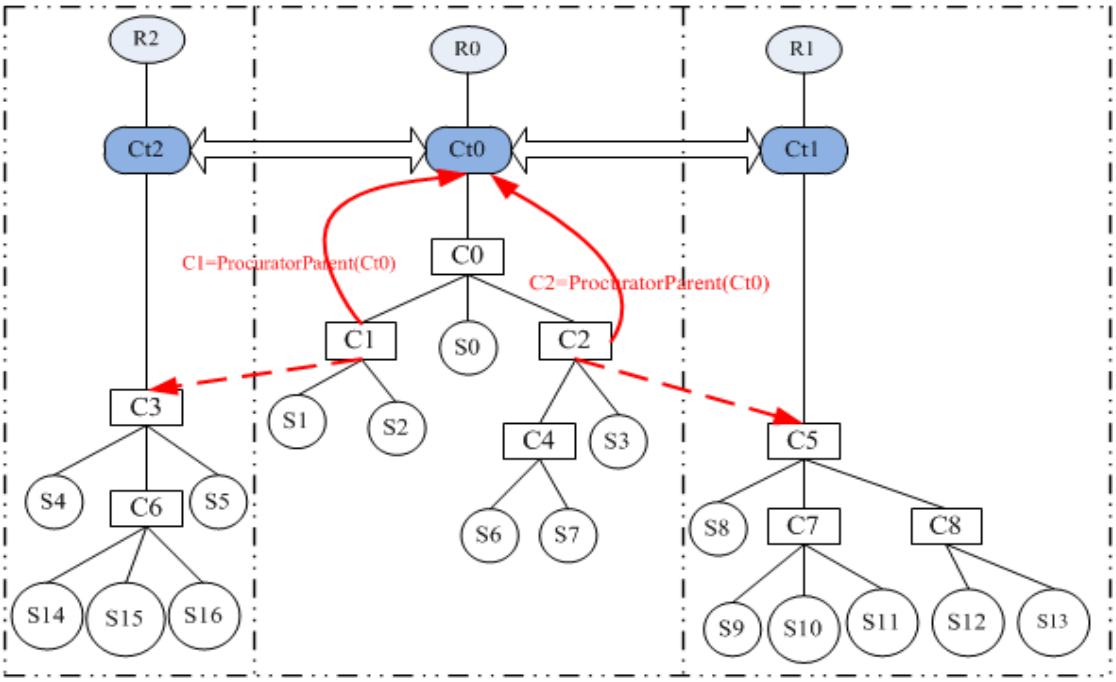


Figure 34. Parent procurateur d'un manteau

4. Protocole de communication

Les types de messages échangés entre les nœuds de base (simulateur, coordinateur et racine) sont les mêmes que ceux définis dans [Zeigler et al. 2000], à savoir, i-message, *-message, x-message, y-message, et D-message. Les nouveaux types de message, introduits pour la SPD avec DEVS ne circulent qu'entre les nouveaux manteaux. Les messages à destination des nœuds (père ou fils) distants sont envoyés au manteau du nœud expéditeur. Cette approche nous permet d'éviter que de nouveaux types de messages apparaissent dans les composants existants.

Les manteaux échangent entre eux des messages de types $(\hat{x}, t)$ et $(\hat{y}, t)$, en plus des messages de synchronisation tels que null-message pour la synchronisation pessimiste, et anti-message dans le cas optimiste. Ces deux nouveaux types de message ont les rôles suivants :

- Les messages (x, t) d'un nœud à destination des fils distants sont envoyés au manteau de son arbre. Ce dernier envoie ensuite $(\hat{x}, t)$ à son fils virtuel correspondant, qui à son tour envoie (x, t) à son fils réel. La figure 35 montre l'exemple de l'envoi (x, t) de C1 à son fils distant C3. Le message est envoyé à Ct0 (qui devient son fils par procuration) sans passer par C0 qui est son parent dans la hiérarchie de base. Ct0 envoie ensuite $(\hat{x}, t)$ à son fils virtuel Ct2 et enfin Ct2 envoie (x, t) à C3.
- Les messages de type (y, t) d'un nœud à destination de son parent distant sont aussi envoyés au manteau de son arbre. A la réception de (y, t), le manteau envoie $(\hat{y}, t)$ à son parent virtuel, qui à son tour envoie (y, t) au fils réel. L'envoi de (y, t) de C5 à son parent distant C2 de la figure 36 illustre cela. (y, t) est envoyé à Ct1 qui devient le

parent par procuration de C5. Ct1 envoie $(\hat{y}, t)$ à son parent virtuel Ct0 et Ct0 envoie (y, t) directement à C2 sans passé par C0 qui est, à la fois, son fils réel et le parent de C2 dans la hiérarchie de base.

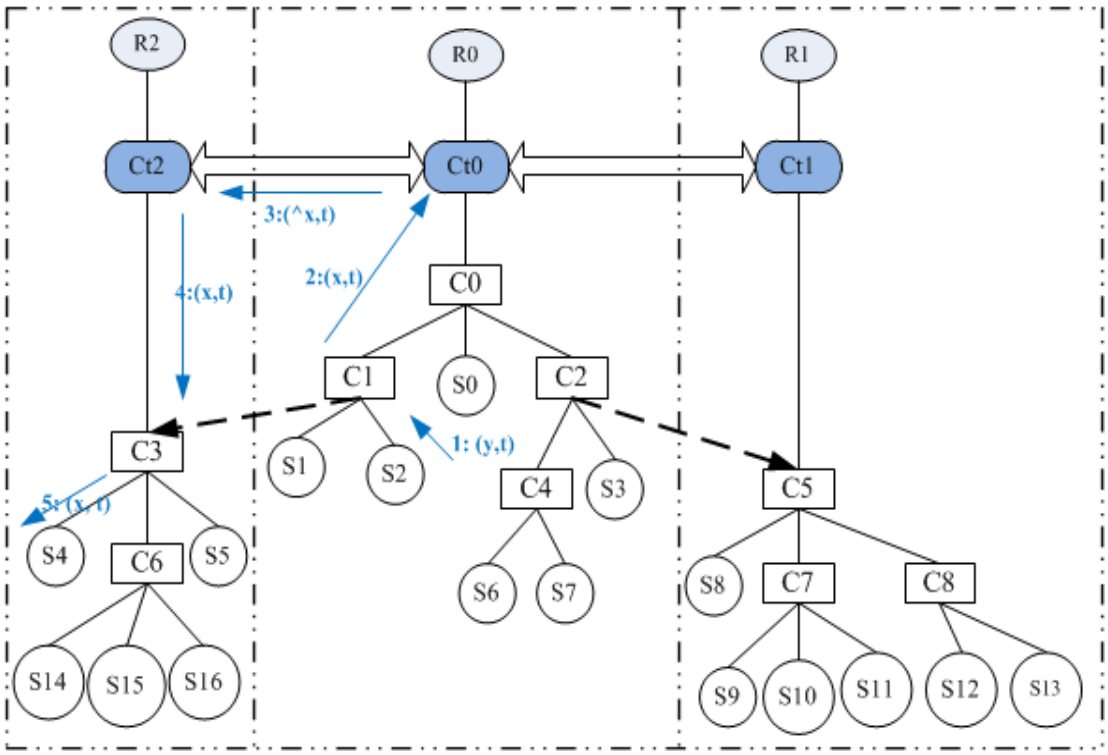


Figure 35. Messages de type $(\hat{x}, t)$

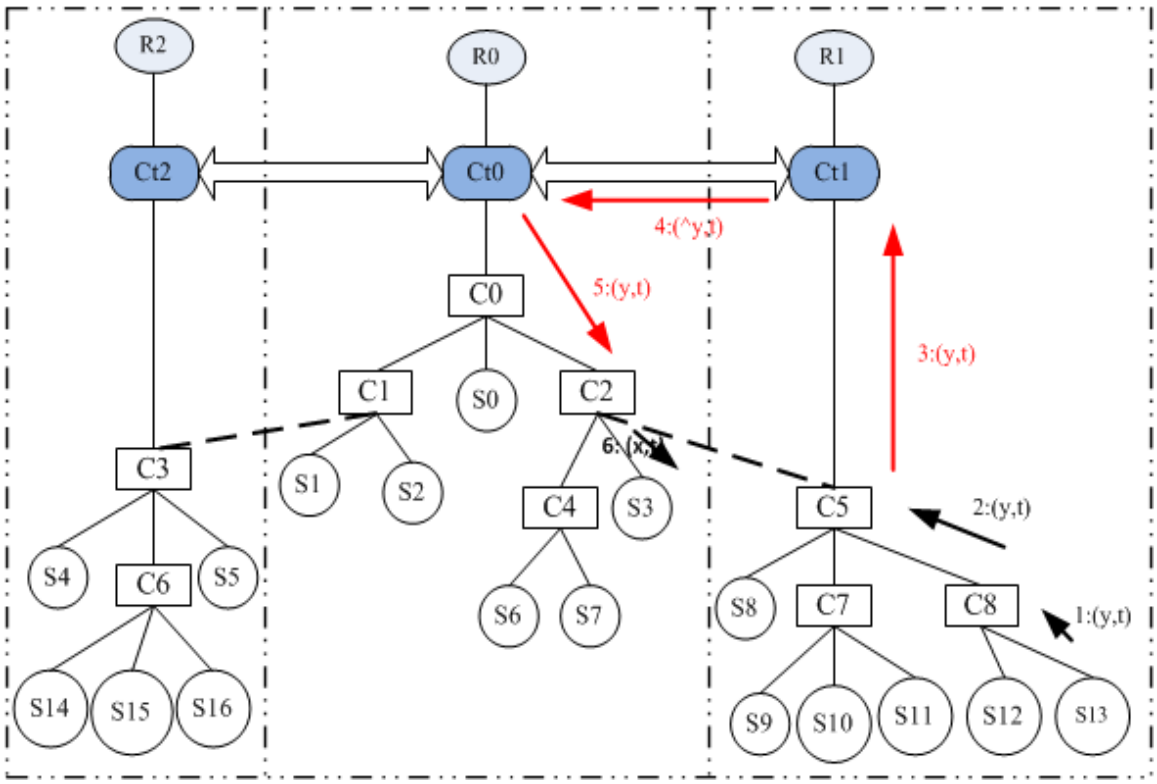


Figure 36. Messages de type $(\hat{y}, t)$

5. Algorithme pour manteau pessimiste

Tous les mécanismes de synchronisation (pessimistes et optimistes) sont implémentés au niveau du manteau. Dans le cas de l'approche pessimiste, il calcule le lookahead et envoie les messages nuls. Pour l'approche optimiste, il sauvegarde les états du sous arbre qu'il contrôle, fait le roll back sur l'ensemble du sous arbre s'il reçoit un roll back-message et envoie des anti-messages.

Pour vérifier le caractère opérationnel de notre approche, nous nous sommes intéressés plus spécifiquement au manteau qui implémente l'approche pessimiste avec message nul. Nous fournissons en Figure 37 l'algorithme pour ce manteau :

- A la réception d'un message de type (*, t) de son parent réel, le manteau envoie (*, t) au fils réel.
- Si le manteau reçoit un message (x, t) de son fils réel, il envoie ($\hat{x}, t$) aux fils virtuels correspondants.
- Si un message (y, t) est reçu par un manteau, il envoie ($\hat{y}, t$) à son parent virtuel.
- Si le manteau reçoit un autre type de message, il le stocke dans la file d'entrée correspondante. Ensuite, il vérifie si toutes les files d'entrée contiennent au moins un message. Si oui, il prélève le message qui a la plus petite date t. Si le message prélevé est de type ($\hat{x}, t$) alors le manteau envoie (x, t) au fils réel destinataire. S'il est de type ($\hat{y}, t$), le manteau envoie (y, t) au parent procurateur correspondant. S'il est de type (null, t) alors le manteau envoie (null, t) à son parent réel.
- Le manteau calcule ensuite le lookahead qu'il propage ses fils et à son parent virtuels via un message de type (null, t). Le lookahead d'un manteau décrit donc le temps mis par ses parents et fils virtuels pour s'attendre à un message en provenance de ce manteau. Le lookahead L est calculé après la consommation d'un message provenant de ses nœuds virtuels (parent et fils), et est envoyé à ces derniers à travers un message nul. Le manteau calcul le lookahead comme défini dans la formule ci-après, et l'envoie à tous ses fils virtuels qui n'ont pas reçu un $\hat{x}$-message de sa part et au parent virtuel s'il n'a pas reçu un $\hat{y}$-message.

$$L = Min(t_{pr}, t_N)$$

- Où t_{pr} est le temps du prochain message d'entrée en attente (nul ou non),

$$t_{pr} = \min_{1 \le i \le n} (t_i) \quad avec\ n = \#(fils\ virtuels) + 1$$

- t_N est le tn du fils réel fourni par un null-message

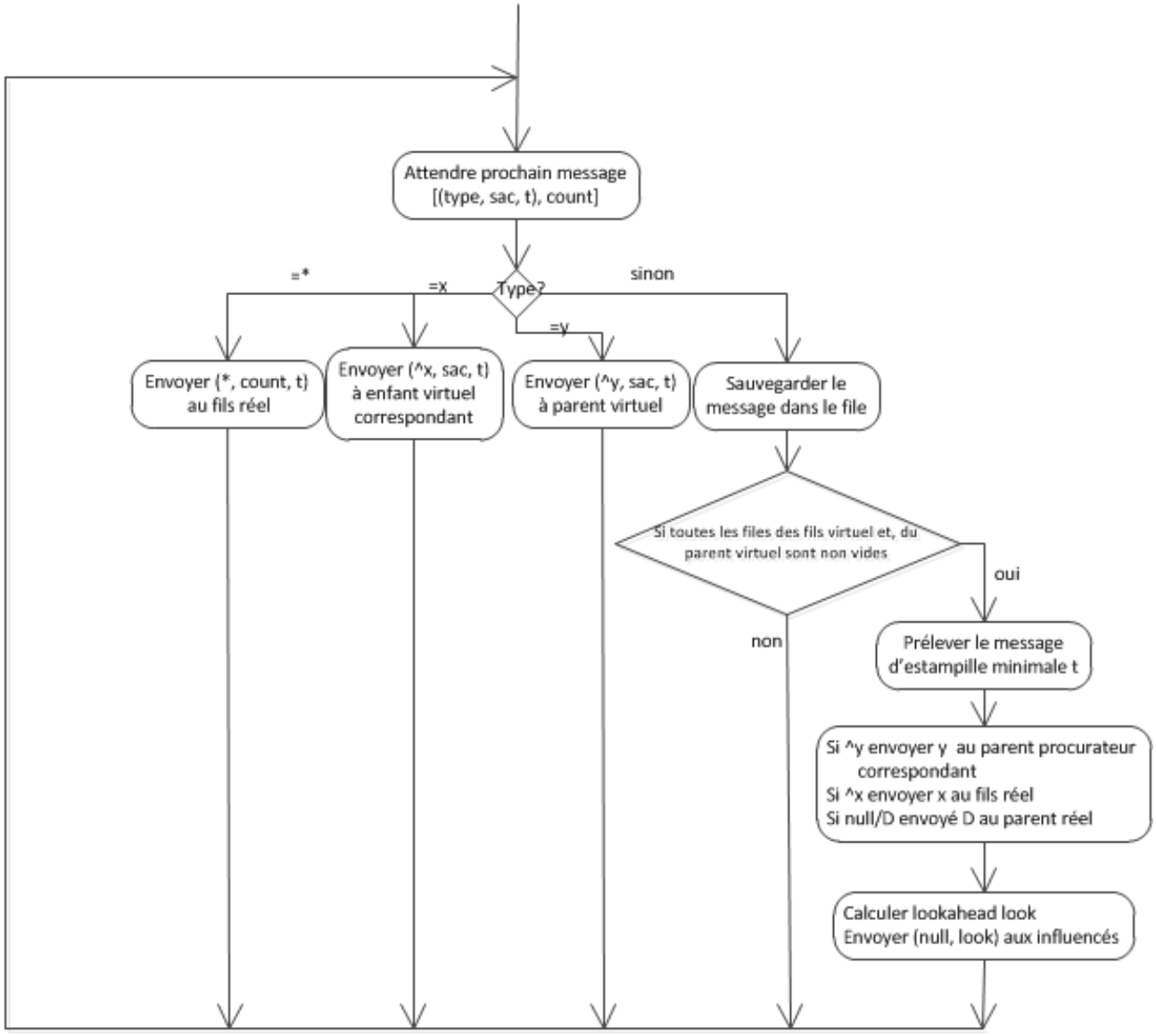


Figure 37. Algorithme du manteau pessimiste

6. Etude de performances

Comme application grande échelle, nous avons construit et effectué des expérimentations de simulation répartie sur un modèle de trafic urbain de la commune 5 du district de Bamako.

La ville de Bamako (Figure 38), érigée en district en 1977 et subdivisée en six communes urbaines, est la capitale du Mali. Elle compte environ 2.000.000 d'habitants (chiffres de 2011). La croissance urbaine de cette ville est la plus élevée en Afrique et la sixième au monde. Bamako est traversée par le fleuve Niger (et partagée presque en deux parties de même superficie). Les rives droite et gauche de la ville sont reliées entre elles par trois ponts.

Notre approche de modélisation a consisté à modéliser d'abord le domaine d'application et à proposer ainsi un méta-modèle pour le trafic urbain. Les éléments dynamiques sont ensuite identifiés dans ce méta-modèle et spécifiés de façon générique sous forme de modèles DEVS (atomiques et couplés). A partir de ce méta-modèle, des modèles concrets sont ensuite

construits, dont celui de la ville de Bamako, mais surtout celui de la Commune 5 de cette ville, que nous utilisons dans ce travail comme banc d'expérimentation (l'effort plus large de modélisation et de simulation de trafic urbain, avec application à la ville de Bamako, fait partie du travail de thèse en cours de Youssouf Koné [Koné 2015]).

Figure 38. Ville de Bamako (Google Maps)

6.1. Méta-modèle de trafic urbain

Le méta-modèle de la Figure 39 vise à fournir une vue générique permettant, par instanciation, de construire de manière hiérarchique et modulaire des modèles plus complexes à partir de modèles de base intervenants dans le trafic.

Le but ici, n'est pas de rentrer dans tous les détails de modélisation. Toutefois, quelques éléments méritent une attention particulière car leur impact est important sur la complexité des modèles élaborés, donc sur les résultats de simulation et les performances d'exécution.

6.1.1.1. Route

Une route (Road) est composée de $n \in \mathbb{N}^*$ lignes (voies). Chaque ligne peut être occupée au même moment par au plus $m \in \mathbb{N}^*$ voitures en fonction de la longueur de la voie. Chaque ligne est divisée en cellules. Ce découpage nous donne une discrétisation de l'espace de type automate cellulaire. Une Route est principalement caractérisée par le nombre de places et l'état d'occupation de ces places. Une place correspond à une cellule de la voie ; elle est caractérisée par son contenu, i.e., le véhicule se trouvant à cette place, son statut, son voisinage et sa longueur. Les véhicules et les interactions entre eux sont les éléments centraux de notre description microscopique. Les interactions sont gérées au niveau de la route.

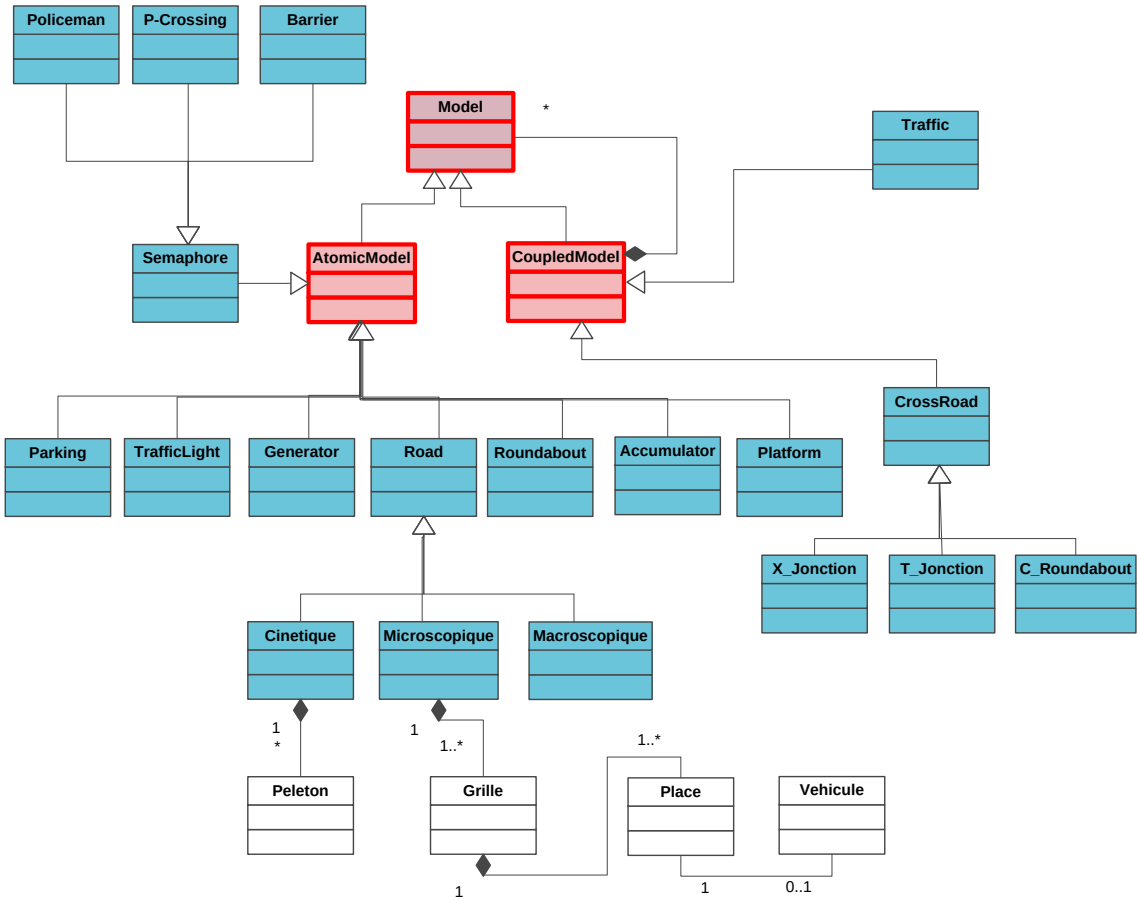


Figure 39. Méta-modèle de trafic urbain

6.1.1.2. Véhicule

Un *véhicule* (*Vehicle*) est un élément qui circule sur le réseau routier. Selon [Gipps 1981], les vitesses maximales que peut prendre un véhicule dans un intervalle de temps $[t, t+T]$, respectivement sans et avec un autre véhicule devant lui sont données comme suit :

$$V_a(n,t+T) = V(n,t) + 2{,}5a(n)T\left(1 - \frac{V(n,t)}{V^*(n)}\right)\sqrt{0{,}025 + \frac{V(n,t)}{V^*(n)}}, \text{ et}$$

$$V_b(n,t+T) = d(n)T + \sqrt{d(n)^2T^2 - d(n)\left[2\big(x(n-1,t) - s(n-1) - x(n,t)\big) - V(n,t)T - \frac{V(n-1,t)^2}{d'(n-1)}\right]}, \text{ où}$$

- $V(n,t)$ est la vitesse du véhicule n à la date t ;
- $V^*(n)$ est la vitesse désirée du véhicule n ;
- $a(n)$ est l'accélération maximale du véhicule n ;
- T est le temps de réaction.
- $d(n) < 0$ est la décélération maximale désirée par le véhicule n ;
- $x(n,t)$ est la position du véhicule n à la date t ;
- $x(n-1,t)$ est la position du véhicule $n-1$ à la date t ;
- $x(n,t+T) = x(n,t) + V(n,t+T)T$
- $s(n-1)$ est la longueur du véhicule $n-1$;
- $d'(n-1)$ est une estimation de la décélération du véhicule $n-1$.

Les interactions entre véhicules se traduisent par des règles d'évolution de l'état de l'espace routier et des positions des véhicules. Les règles sont appliquées au niveau de chaque place en tenant compte des états des places voisines.

La 1ère règle concerne les conditions pour un véhicule d'en doubler un autre sur sa gauche. Considérons les positions des véhicules C_1, C_2 et les leurs voisins présentés sur la Figure 40. Si la vitesse de C_1 est supérieure à celle de C_2 et que la distance entre les deux est supérieure ou égale à la distance minimum nécessaire pour le doublage à gauche et que les distances minimales entre le premier véhicule se trouvant devant lui à gauche et le premier véhicule se trouvant derrière lui à gauche sont garanties alors C_1 peut doubler C_2 en allant vers la gauche. Après le doublage, C_1 peut changer de vitesse. La notion de distance peut être remplacée par le nombre de places vides entre véhicules sans perte de généralité.

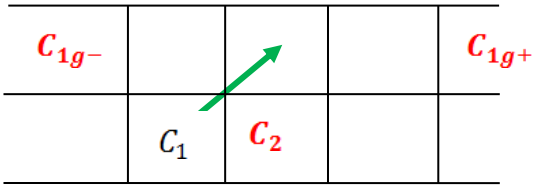


Figure 40. Figure. Exemple de voisinage entre véhicules

Soit C un véhicule, et notons :

- $ligne(C)$, la ligne sur laquelle se trouve C ;
- $colonne(C)$, la colonne sur laquelle se trouve C ;
- v_C, la vitesse de C ;
- $Leader, Leader_g$ et $Leader_d$ sont respectivement les leaders directement devant, à gauche et à droite ;
- $Suiveur$, $Suiveur_g$ et $Suiveur_d$ sont respectivement les suiveurs directement derrière, à gauche et à droite ;
- D_0 désigne la distance minimale entre deux véhicules ;
- $D_l(v_C)$ désigne le seuil de distance entre véhicule C et son leader en fonction de sa vitesse ;
- $D_l^c(v_C)$ désigne le seuil de distance entre véhicule C et son leader ou son suiveur de gauche ou de droite en fonction de sa vitesse ;
- $D_{Ra}(v_C)$ désigne le seuil de ralentissement ;
- $D_A(v_C)$ désigne le seuil d'accélération.

Les conditions de doublage d'un véhicule en allant vers la gauche sont :

- 1er cas : le leader de C existe
  - ✓ $ligne(C) \neq 0$ (le véhicule C se trouve pas déjà sur la ligne située à l'extrême gauche auquel cas il n'est pas possible de changer de ligne vers la gauche)
  - ✓ $v_C > v_{Leader}$ (C va plus vite que son leader, il a donc le choix de ralentir ou de doubler)
  - ✓ $D_0 \leq x_{Leader} - x_C \leq D_l(v_C)$ (le seuil de distance entre C et son leader est franchi et la distance minimale est garantie)
  - ✓ $Leader_g = null \vee (Leader_g \neq null \wedge x_{Leader_g} - x_C > D_l^c(v_C))$ (soit il n'y a pas de véhicule devant lui à gauche ou il possède un leader à gauche se trouvant à une distance suffisante)
  - ✓ $Suiveur_g = null \vee (Suiveur_g \neq null \wedge x_C - x_{Suiveur_g} > D_l^c(v_{Suiveur_g}))$

- 2ème cas : le leader de C n'existe pas
  - ✓ $Leader_g = null \vee (Leader_g \neq null \wedge x_{Leader_g} - x_C > D_l^c(v_C))$
  - ✓ $Suiveur_g = null \vee (Suiveur_g \neq null \wedge x_C - x_{Suiveur_g} > D_l^c(v_{Suiveur_g}))$

Lorsque ces conditions sont satisfaites, alors C peut doubler à gauche. Dans ce cas les nouvelles valeurs des attributs de C sont calculées de la façon suivante :

- $ligne(C) = ligne(C) - 1$
- $colonne(C) = colonne(C) + 1$
- $Place(C) = Places[ligne(C) - 1][Colonne(C) + 1]$
- $v'_C = \begin{cases} \widetilde{v_C} & si\ x_{Leader_g} - x_C > D_f(v_C) \vee Leader_g = null \\ v_C & sinon \end{cases}$ (C accélère ou conserve sa vitesse) où $\widetilde{v_C}$ est une valeur choisie aléatoirement dans l'intervalle $[v_c, V_{max}]$.
- $v'_{Suiver} = \begin{cases} \widetilde{v_{Suiveur}} & si\ x_{Leader} - x_{Suiveur} > D_f(v_{Suiveur}) \\ v_{Suiveur} & sinon \end{cases}$ (le suiveur de C peut accélérer ou conserver sa vitesse en fonction de la distance existant entre lui et leader de C).
- $Leader = Leader_g\ si\ Leader_g \neq null$
- $t_g = \frac{d}{v_C}$ si la place de gauche n'est pas encombrée, $t_g = +\infty$ sinon
- $t_d = t_a = +\infty$ (temps restant pour aller dans la cellule de devant et de droite sont infinis car le choix d'aller à gauche a été fait)
- $x_C = x_C + ev_C$ (e le temps de parcours depuis le dernier changement)

Les conditions de doublage à droite et les opérations de mise à jour des variables sont définies de façon symétrique au cas précédent (les conditions de doublage à gauche sont prioritaires sur celles de doublage à droite).

Pour qu'un véhicule ralentisse, les conditions suivantes sont nécessaires :

- 1er cas : le leader de C existe
  - ✓ $v_C > v_{Leader}$ (C va plus vite que son leader, il peut donc ralentir ou doubler) ;
  - ✓ $D_0 \leq x_{Leader} - x_C \leq D_{Ra}(v_C)$ (le seuil de ralentissement entre C et son leader est franchi et la distance minimale est garantie).
- 2ème cas : le leader de C n'existe pas
  - ✓ $x_{Max} - x_C \leq D_{Ra}(v_C)$ (le seuil de ralentissement entre C et la position du feu est franchi).

Dans ce cas C adopte une nouvelle vitesse inférieure à sa vitesse actuelle. On a par exemple $v'_C \in]0, v_C[$ et $v'_C = [\beta + (1-\beta)\xi]v_C$ où ξ est distribué uniformément sur l'intervalle $[0,1]$ et $\beta < 1$. Après le ralentissement, le véhicule doit avoir la possibilité d'accélérer à nouveau. Cela est exprimé par la contrainte suivante : $D_A(v'_C) > D_{Ra}(v_C)$.

Pour qu'un véhicule accélère, les conditions suivantes sont nécessaires :

- $v_C < v_{Leader}$ (C va plus vite que son leader, il a donc le choix de ralentir ou doubler)
- $D_0 \leq x_{Leader} - x_C \geq D_A(v_C)$ (le seuil d'accélération entre C et son leader est atteint et la distance minimale est garantie)

Dans ce cas, le véhicule C adopte une nouvelle vitesse supérieure à sa vitesse actuelle. Par exemple on a : $v'_C \in]v_C, V_{max}[$ et $v'_C = v_C + \xi(\min(V_{max}, \alpha v_C) - v_C)$ où ξ est distribué

uniformément sur l'intervalle [0,1] et $\alpha > 1$. Après l'accélération, le véhicule doit pouvoir ralentir à nouveau. Cela est exprimé par la contrainte suivante : $D_A(v_C) > D_{Ra}(v'_C)$.

6.1.1.3. Autres composants

Les *feux tricolores* (*TrafficLight*) permettent de réguler la circulation aux intersections. Un feu tricolore affiche successivement trois couleurs (vert, orange, rouge) indiquant aux véhicules les droits/obligations respectifs de passer, de ralentir et de s'arrêter. Les *sémaphores* (*Semaphore*) sont des alternatives aux feux tricolores pour la régulation de trafic. Les *générateurs* (*Generator*) sont les composants qui modélisent les arrivées de véhicule dans le trafic, alors que les accumulateurs (*Accumulator*) sont des composants qui récupèrent les voitures sortant de la circulation (permettant ainsi des calculs de statistiques sur les temps de séjour des véhicules dans le trafic).

6.2. Modèle de la Commune V

Le modèle de plus haut niveau est celui de la ville de Bamako (Figure 41). C'est un couplage des modèles des rives gauche et droite RD et RG), reliées par les modèles des 3 ponts. La Commune V est un composant du modèle de la rive droite (Figure 42). Il est alimenté par (et dessert) d'autres quartiers. Le cheminement des véhicules est défini en fonction des densités supposées de ces quartiers. Le modèle de la commune V (Figure 43) est un couplage des modèles des quartiers (Kalaban-coura, Baco-Djicoroni, Torokorobougou, Quartier Mali, Badalabougou et Daoudabougou). Chaque modèle de quartier est un couplage de modèles de route et de modèles de ronds-points. Chaque rond-point à son tour est un couplage de places.

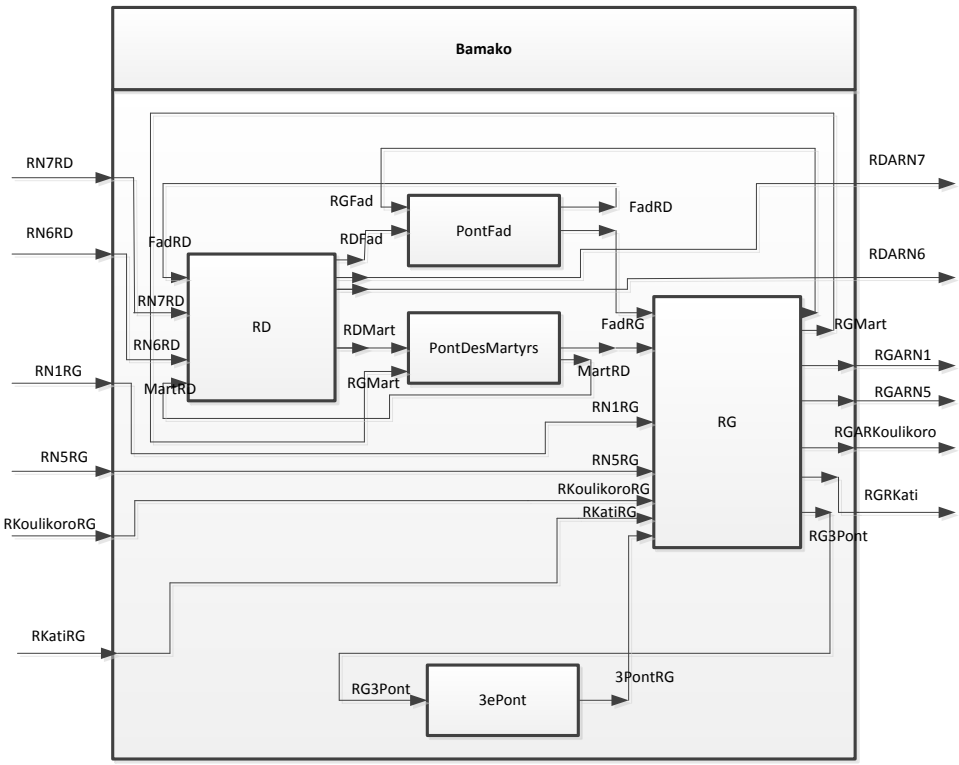


Figure 41. Modèle DEVS du trafic urbain de la ville de Bamako

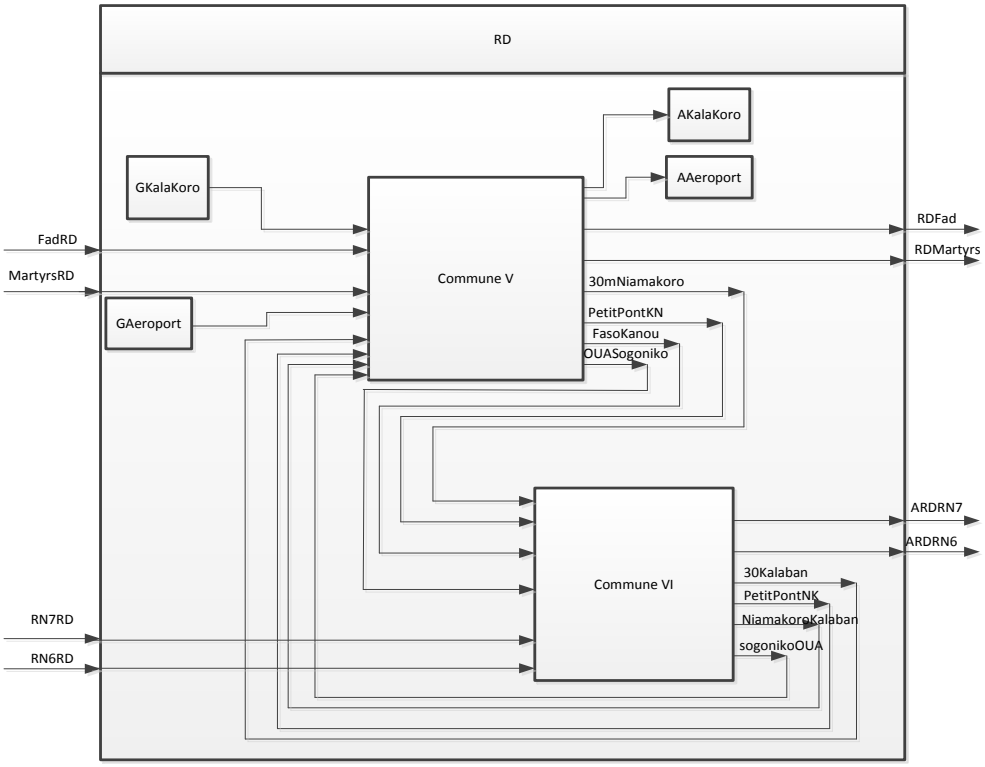


Figure 42. Modèle DEVS de la Rive Droite

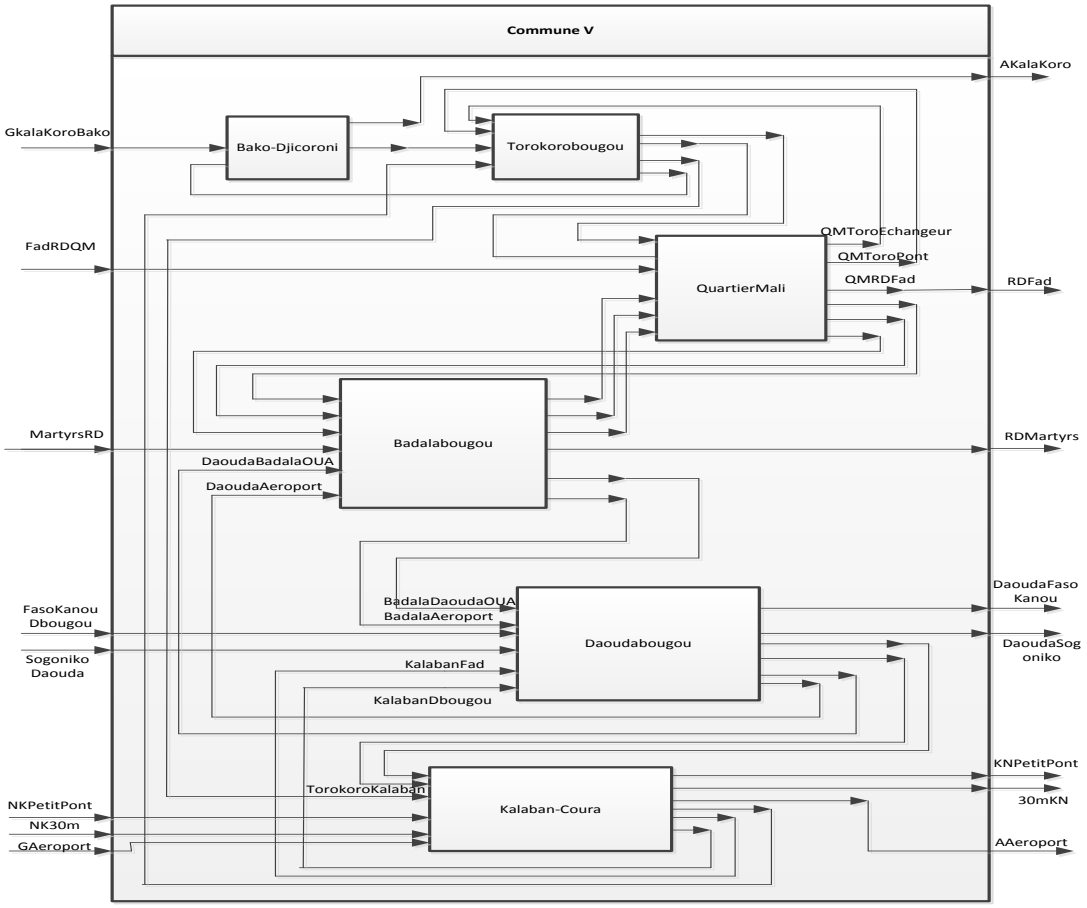


Figure 43. Modèle DEVS de la commune 5.

6.2.1. *Graphe de simulation du trafic de la commune V*

L'arbre de simulation de ce modèle est donné en Figure 44. Pour son partitionnement, nous avons utilisé l'algorithme de partitionnement hiérarchique HIPART [Kim et al. 1998]. La Figure 44 indique par un jeu de couleurs le résultat de ce partitionnement en quatre parties. Le graphe de simulation correspondant, par injection de manteaux, est donné par la Figure 45.

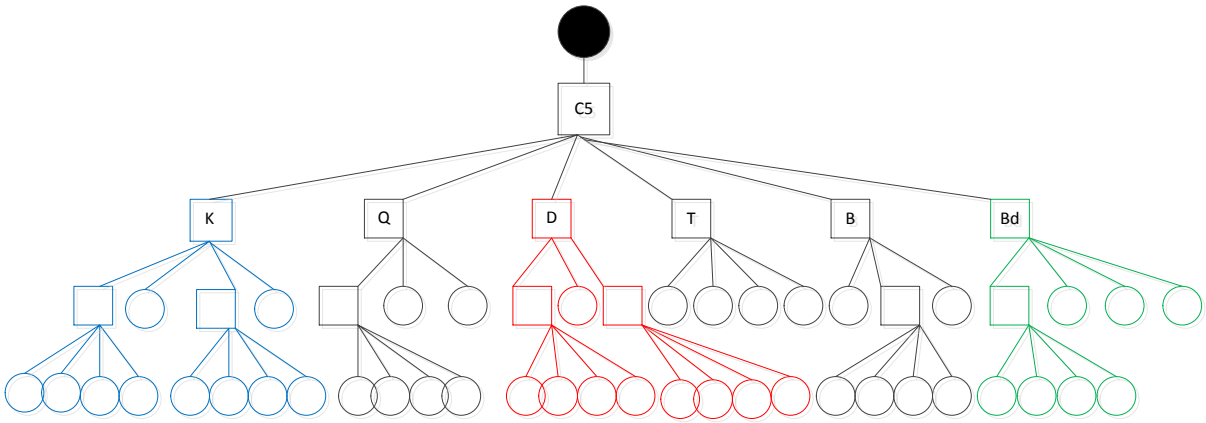


Figure 44. Arbre de simulation du modèle de la commune V

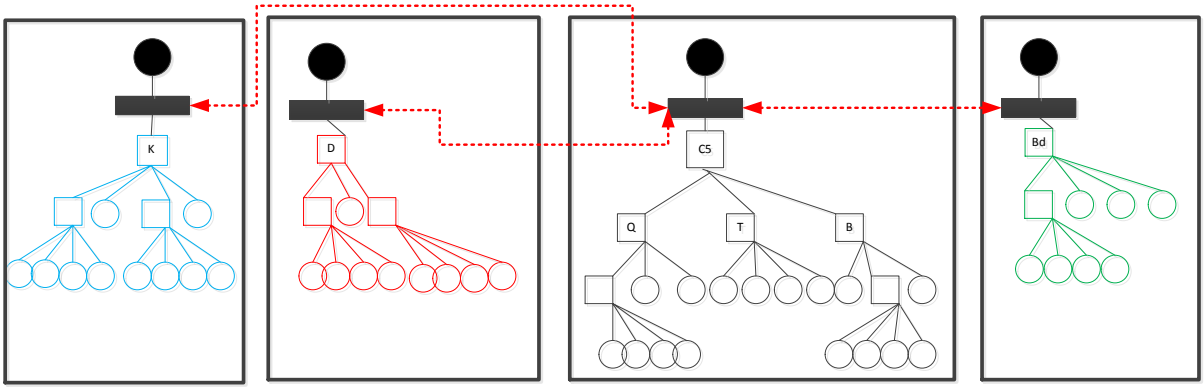


Figure 45. Graphe de simulation

6.2.2. *Communications inter processeurs*

Nous avons envisagé, en plus de l'exécution de la version séquentielle du modèle DEVS de la commune V, 2 solutions de communication inter processeurs pour la simulation répartie du même modèle :

- L'exécution parallèle sur une Machine Virtuelle MPI construite par-dessus 4 des 20 PCs identiques connectés en réseau filaire Ethernet dans une salle dédiée.
- L'exécution distribuée sur 4 autres PCs du même groupe de 20 PCs, mais avec une solution CORBA déployée entre les processeurs, pris deux par deux.

6.2.2.1. Solution MPI

Nous avons utilisé l'API libre P2P-MPI [Genaud & Rattanapoka 2007] pour créer une Machine Virtuelle. Cet API vise à fournir un moyen d'exécuter des programmes Java parallèles sur un ensemble d'ordinateurs interconnectés. La bibliothèque MPJ de P2P-MPI

permet de créer un multiprocesseur virtuel (appelé VM pour Machine Virtuel) par-dessus un réseau d'ordinateurs (à condition que ce dernier soit ouvert, i.e., sans pare-feu ni antivirus), d'ajouter à la MV les références des CPUs que l'installation arrive à reconnaître dans le réseau, de démarrer (ou d'arrêter) la MV, et de lancer plusieurs processus sur cette MV. Il existe deux modes d'exécution : (1) un mode désigné, où l'utilisateur affecte un processus à une CPU de la MV, et (2) le mode transparent, où c'est la MV qui s'occupe des affectations de processus, et de la répartition de charge dynamique.

6.2.2.2. Solution CORBA

Pour la solution CORBA, nous avons utilisé le schéma de RMI-CORBA de Java, avec une légère adaptation à la nécessité que les envois de message puissent se faire dans les deux sens, entre processeurs. Ainsi, dans le schéma Client-Serveur que décrit la Figure 46, chaque processeur est à la fois client et serveur. C'est pourquoi, le Client implémente le même contrat que le Serveur, qui consiste en :

- Une méthode pour permettre à un tiers de se connecter. L'appel distant de la méthode *connect()* d'un tiers A par un tiers B, entraîne l'enregistrement du nom du tiers B dans une variable locale du tiers A comme un de ses fils ou parents virtuels. Ceci permet ultérieurement au tiers A d'invoquer la méthode distante *warn()* du tiers B.
- Une méthode pour permettre à un tiers d'être alerté de l'arrivée d'un message distant. Lors de l'appel à la méthode *warn()*, le message est passé en paramètre.

Chaque composant s'inscrit (avec son nom de composant) dans le registre RMI par la méthode *bind()*, et récupère dans ce même registre, par leurs noms, les références réseau des autres composants par la méthode *lookup()*.

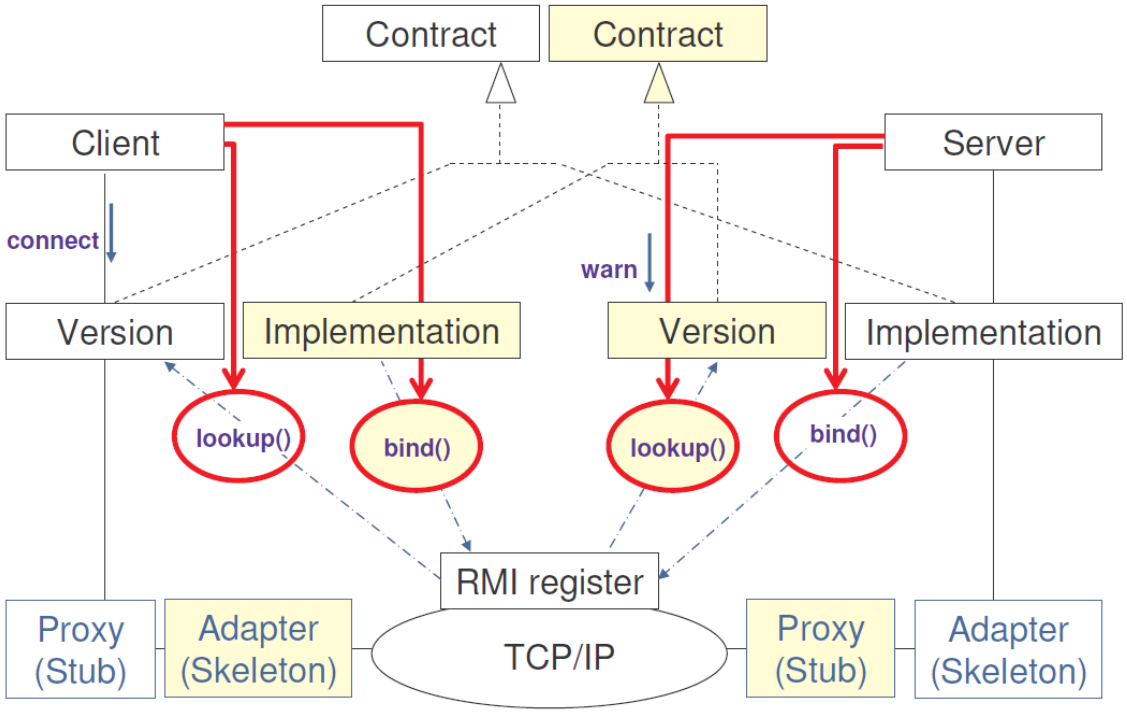


Figure 46. Schéma de communication CORBA entre processeurs

6.3. Expérimentations

Comme indiqué précédemment, nous avons utilisé une salle de 20 PCs identiques (Intel Core i5, 1,8 Ghz, 64 bits, 16 Go de RAM) sous Linux, tous connectés en réseau filaire Ethernet. Nous avons initié des sessions de simulation de respectivement 10.000 unités de temps (u.t.), 100.000 u.t. et 1.000.000 u.t., dont les résultats sont présentés respectivement par les Figures 47, 48 et 49. Pour chacun de ces cas, nous avons effectué 10 réplications de simulation. Les durées d'exécution sont enregistrées en secondes, en utilisant les horloges systèmes des PCs. Dans le cas réparti, la durée d'exécution d'une session de simulation est la durée d'exécution du processus le plus long (en d'autres termes, la simulation est considérée terminée, quand le dernier processus s'arrête). En l'absence de données réelles existantes pour calibrer le modèle, nous avons utilisé des valeurs approximatives basées uniquement sur notre connaissance relative du terrain (le travail de thèse en cours sur ce problème comporte un important volet de collecte de données sur le terrain [Koné 2015]).

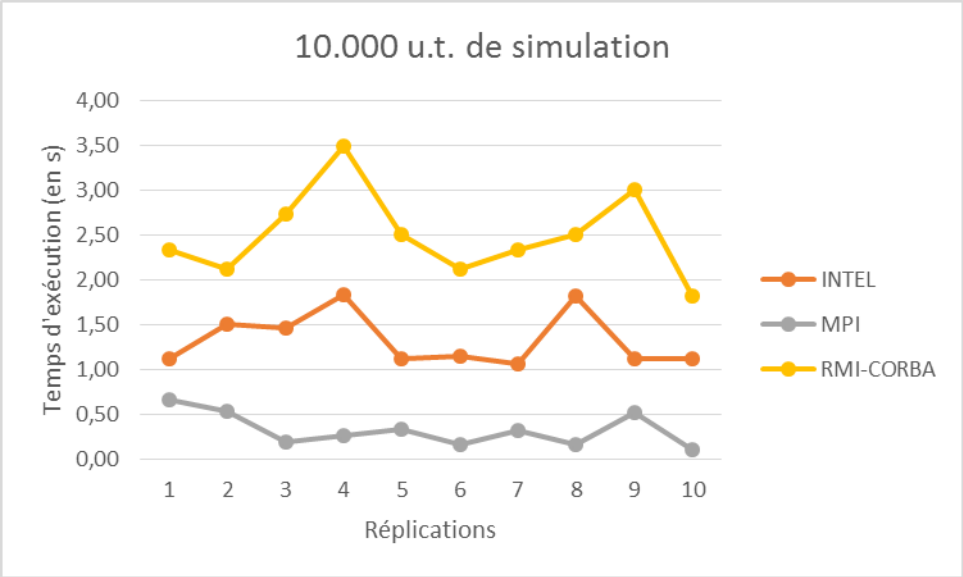


Figure 47. Résultats comparés pour 10.000 u.t. de simulation

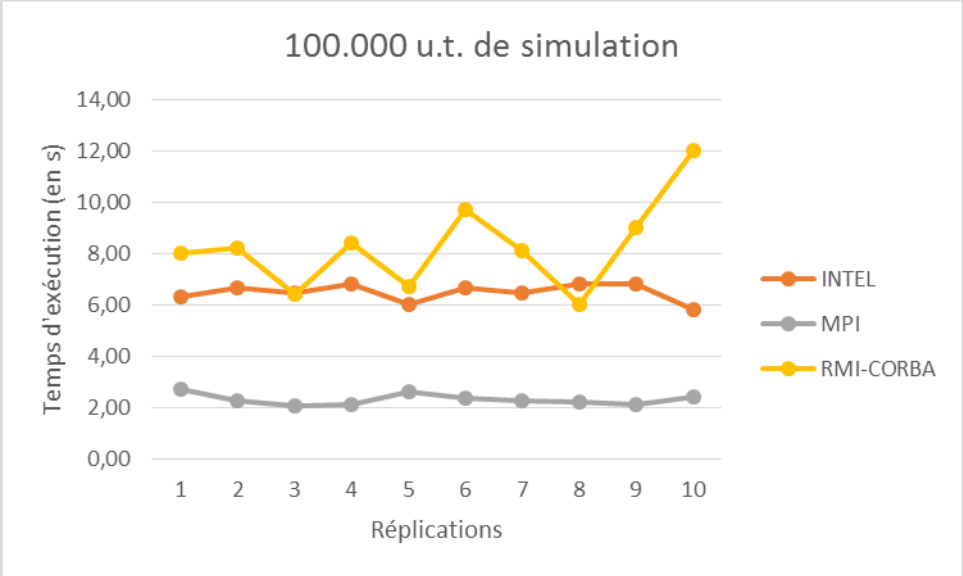


Figure 48. Résultats comparés pour 100.000 u.t. de simulation

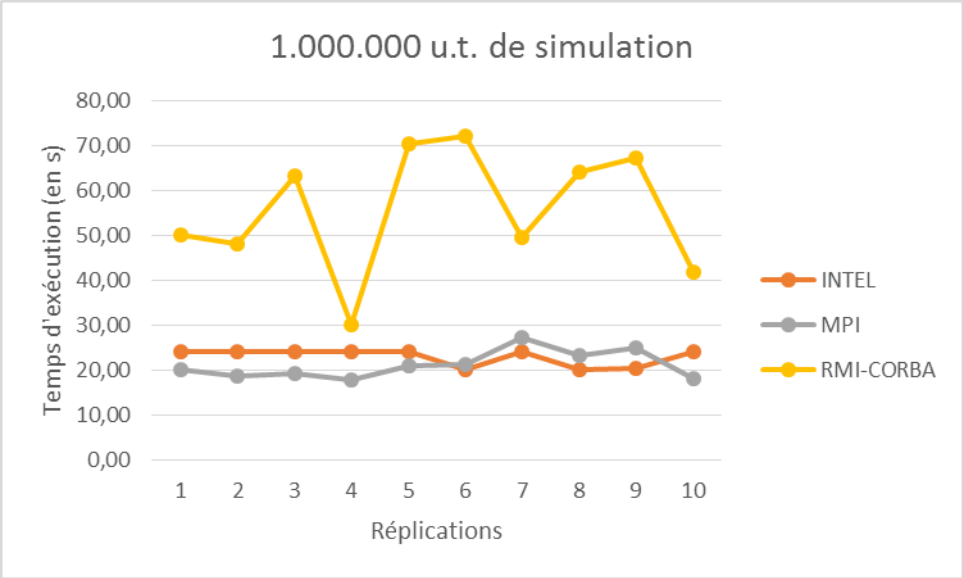


Figure 49. Résultats comparés pour 1.000.000 u.t. de simulation

7. Conclusion

Pour une exécution séquentielle, il existe de nos jours plusieurs implémentations de DEVS (et de ses multiples extensions, telles DSDEVS, DEV&DESS, Cell-DEVS…). Toutes ces implémentations s'appuient sur les algorithmes définis initialement, avec des modifications pour les extensions. Ces implémentations, une fois réalisées, se heurtent au problème de passage à l'échelle. Pour des modèles de très grande taille, le besoin de faire de la simulation répartie s'impose. Toutes les solutions réparties de DEVS dans la littérature montrent des altérations des algorithmes existants, et impliquent donc des efforts d'implémentation non négligeables pour les utilisateurs des packages séquentiels, s'ils ne disposent pas des versions distribuées.

L'idée du travail présenté ici est de proposer une démarche, permettant de mettre en œuvre les approches de SPD avec une implémentation séquentielle DEVS existante, sans être obligé de modifier les algorithmes de ce dernier (donc d'altérer le code en profondeur). A la place, nous adoptons un principe de fusion modulaire entre DEVS et la SPD, qui donne naissance à un graphe de simulation dans lequel plusieurs arbres de simulation séquentielle DEVS sont fédérés grâce à une couche horizontale de SPD. Les éléments de bases de l'arbre de simulation ignorent l'existence de la stratégie de simulation répartie. Nous avons appliqué cette approche à un cas d'étude de simulation de trafic urbain. Les résultats expérimentaux sont prometteurs.

Un des enseignements que nous tirons de cette étude de cas est que l'ingénierie d'un système de simulation DEVS répartie, par injection de manteaux, a toutes les caractéristiques d'une ingénierie dirigée par les modèles, où :

- Le CIM est le modèle DEVS. En ce sens, il est indépendant de tout calcul, car il exprime la solution sous sa forme purement syntaxique.
- Le PIM est le graphe de simulation. Il donne le sens de la solution (sémantique opérationnelle), mais de manière indépendante de toute plateforme.
- Le PDM est la description des mécanismes de communication inter processeurs (Machine Virtuelle PVM, CORBA ou plateforme monoprocesseur).

- Le PSM est la description UML des classes SimStudio instanciées pour implémenter les coordinateurs, simulateurs, racines et manteaux, et qui intègrent celle des classes réalisant les communications inter processeurs.
- Dans le contexte de notre environnement de simulation (appelé SimStudio), le code généré est en Java.

C'est dans cet esprit que nous proposons, dans le chapitre qui suit, une démarche générique d'IDM pour l'ingénierie de la simulation répartie DEVS, visant à systématiser les efforts présentés dans le présent chapitre, afin de rendre aisée sa réutilisation pour d'autres modèles ainsi que pour d'autres implémentations DEVS.

Chapitre IV.
IDM DE SIMULATION DEVS REPARTIE

1. Introduction

Nous avons vu que DEVS est un formalisme de modélisation et simulation qui sépare le modèle du simulateur. Un modèle DEVS est soit atomique (i.e., non décomposable en modèles composants), soit couplé (i.e., composé d'autres modèles DEVS). Un arbre de simulation DEVS est obtenu à partir du modèle DEVS en faisant correspondre, en plus du coordinateur racine qui gère l'ensemble de la simulation, un simulateur à un modèle atomique et un coordinateur à un modèle couplé. Cet arbre est destiné à une exécution séquentielle de la simulation. Pour une exécution répartie, l'arbre de simulation doit être projeté sur un graphe de simulation. La génération d'arbre ou de graphe est un acte répété par les implémentations de DEVS (qu'elles soient séquentielles ou réparties) chaque fois qu'un nouveau modèle DEVS est produit. La Figure 50 illustre ce concept.

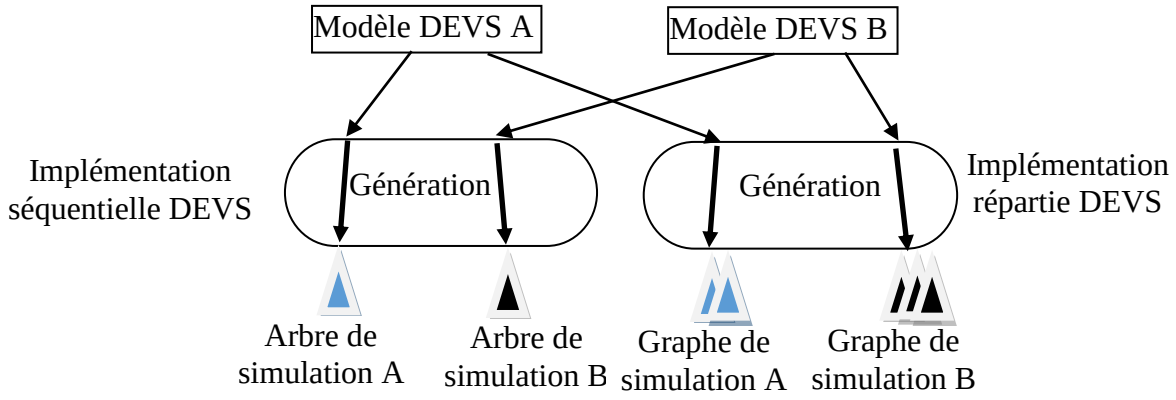


Figure 50. Logique d'implémentation de DEVS

Dans le chapitre précédent, nous avons proposé une approche permettant à une implémentation répartie DEVS de réutiliser les composants d'une implémentation séquentielle sans les altérer. Dans le présent chapitre, nous allons nous intéresser à la systématisation du processus de génération de graphe de simulation à chaque fois qu'un nouveau modèle DEVS est produit.

Notre approche s'inspire de l'Ingénierie Dirigée par les Modèles, et nous proposons une Architecture Dirigée par les Modèles pour l'Ingénierie des systèmes de simulation DEVS répartie (appelée MD3SEA, pour Model Driven Distributed DEVS Simulation Engineering Architecture). A cet effet, la Figure 50 montre que le modèle DEVS s'apparente à la connaissance métier du système (il ne change que si le système change ou que la connaissance sur le système courant change), et ce de manière indépendante de la manière dont ce dernier opère. L'arbre de simulation (dans le cas séquentiel) et le graphe de simulation (dans le cas réparti) expriment la logique métier (ils décrivent le comportement algorithmique du système, sans donner de détails sur l'implémentation et la plateforme d'exécution).

Nous présentons d'abord les principes de MD3SEA (modèle DEVS = CIM, graphe de simulation = PIM, diagramme d'implémentation dans SimStudio = PSM) et nous expliquons l'approche de transformation de modèles adoptée dans cette architecture, ainsi que le processus de transformations successives de modèles, du CIM au PSM. Puis, nous présentons notre Framework conçu pour appliquer les principes de MD3SEA, et qui met en œuvre notre approche de parallélisation de simulateurs DEVS existants.

2. MD3SEA

Notre architecture MD3SEA, comme le montre la Figure 51, fournit un cadre global d'ingénierie de simulation DEVS répartie qui repose sur l'usage des langages DEVS, UML et XML (les DSLs du niveau M2), et intègre (au niveau M1) une méthodologie que nous appelons SGF (pour Simulation Graph Framework) qui guide la modélisation et les transformations successives de modèles, du CIM au PSM. L'application du SGF permet aux utilisateurs de paralléliser leurs simulateurs DEVS séquentiels.

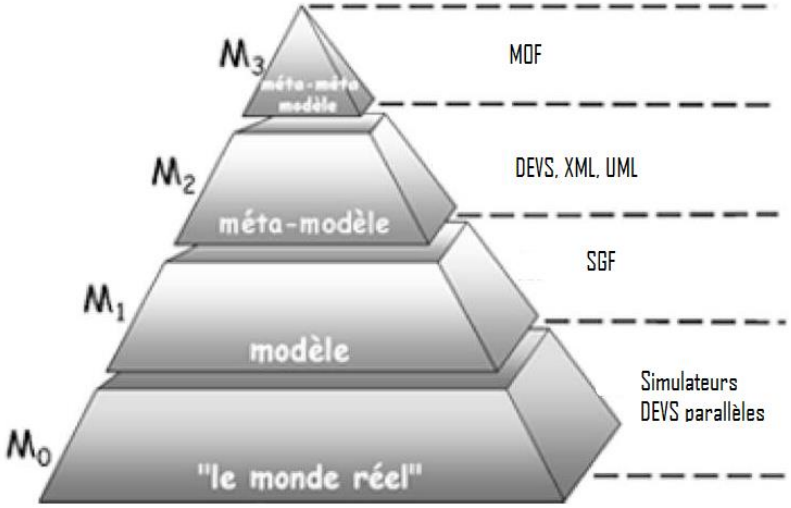


Figure 51. MD3SEA

Toutes les transformations de modèles dans MD3SEA utilisent des représentations XML. Ceci permet de capturer les règles de transformation en utilisant XSLT [W3C 2015]. Ce dernier, appliqué à un fichier XML source, produit un fichier XML cible. Les règles codées dans XLST implémentent les fonctions mathématiques de correspondance entre les ensembles abstraits représentatifs des différentes structures de simulation mise en évidence dans le SGF. La Figure 52 illustre le principe général de transformation de modèle adopté dans MD3SEA.

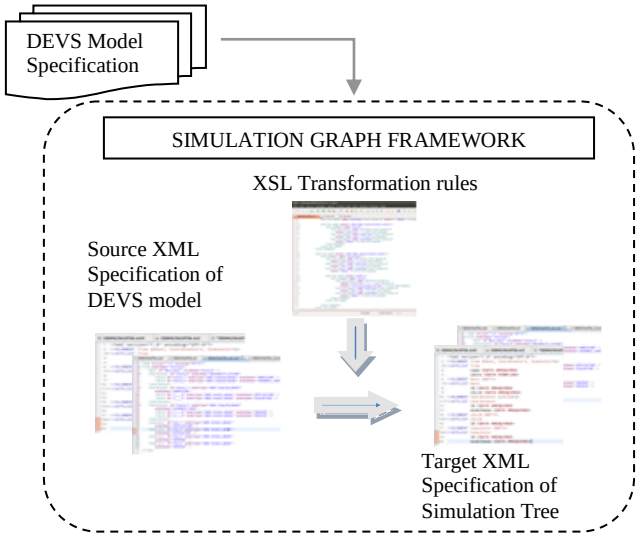


Figure 52. Transformations dans MD3SEA

3. Simulation Graph Framework

Nous avons proposé dans [Adegoke et al. 2011], [Adegoke et al. 2013] une méthodologie permettant de construire un graphe de simulation. C'est une approche multicouches, constituée des différentes structures de données qui interviennent dans la simulation DEVS parallèle et distribuée. Cette construction par niveaux permet d'éviter les erreurs inhérentes à la construction d'un système de simulation DEVS adapté à l'exécution parallèle et distribuée. Il s'agit en fait d'un passage du CIM au PSM et au code, selon le schéma en Y caractéristique du MDA (Figure 53). Après la transformation du CIM en PIM, une série de raffinements sont apportées pour faire progresser ce PIM d'arbre en graphe. A la dernière étape de ce PIM raffiné, une intégration doit être faite avec le PDM (qui décrit les mécanismes de communication entre processeurs, comme CORBA, MPI, EJB, Web services, etc.) pour produire le PSM. Ce dernier sert à générer le code final de simulation DEVS réparti.

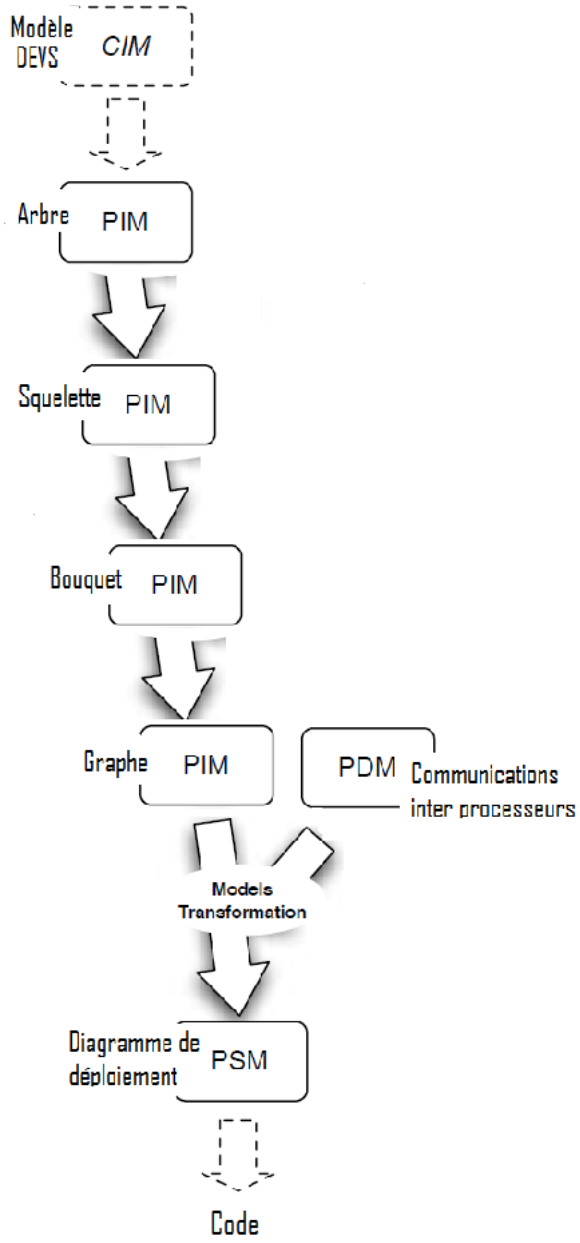


Figure 53. Schéma en Y dans MD3SEA

La Figure 54 donne l'ensemble des chemins possibles qui peuvent être pris pendant la construction d'un graphe de simulation. Cette méthodologie itère sur chaque étape jusqu'à ce que certains critères de satisfaction définis par l'utilisateur soient atteints. Pour passer d'une étape à une autre de la méthodologie, différentes fonctions doivent être définies par l'utilisateur, notamment *Split*, *Cluster*, *Map* et *Transform* (Ces différents processus se poursuivent jusqu'à satisfaction des critères l'utilisateur) :

- A partir de l'arbre de simulation, l'utilisateur crée une partition de nœuds en définissant la fonction *Split*. Ceci produit une structure que nous appelons squelette de simulation (Simulation Skeleton).
- La fonction *Cluster* lui permet de définir le regroupement des nœuds en processus (un nœud pour un processus, ou plusieurs nœuds encapsulés dans un même processus). Nous appelons bouquet de simulation (Simulation Bundle) la structure produite.
- La fonction *Map* (parfois appelée *Distrib*) lui permet de projeter l'ensemble des processus sur l'ensemble des processeurs disponibles.
- La fonction *Transform* lui permet de modifier la structure de l'arbre de simulation, soit par expansion, soit par réduction. Cette fonction modifie non seulement le nombre de nœuds de simulations de l'arbre (simulateurs et coordinateurs) mais aussi les relations entre les nœuds, donc modifie le protocole de simulation DEVS. Ainsi, la méthodologie permet d'aplatir ou de gonfler l'arbre initial avant de passer aux étapes ultérieures de répartition.

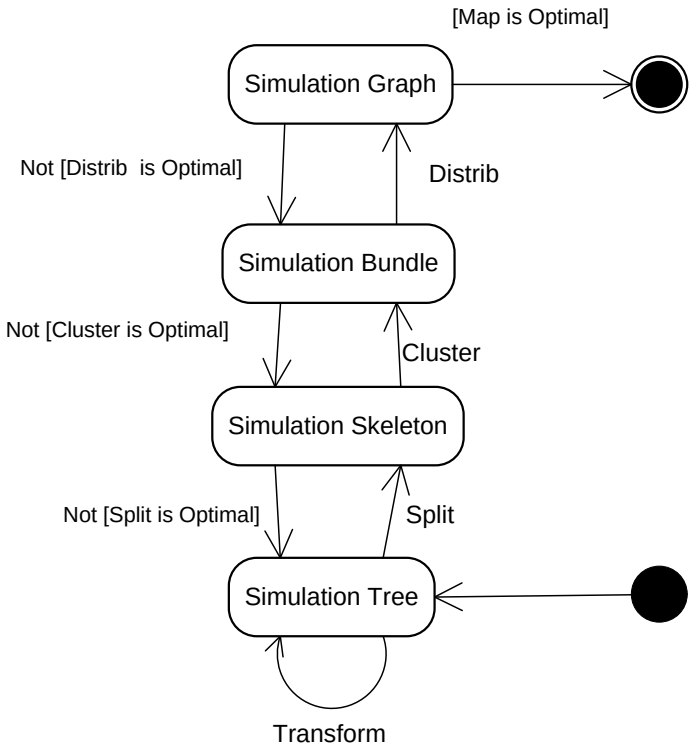


Figure 54. Méthodologie SGF

Il importe de noter que l'approche par manteau, présentée dans le chapitre précédent, n'est qu'un parcours particulier de la méthodologie SGF, avec des fonctions Transform, Split, Cluster et Map spécifiques.

Les sous-sections qui suivent montrent les blocs de construction utilisés dans la description de la représentation XML de chacune des structures de données de simulation. Chacune de ces

descriptions est conforme à notre méthodologie et prend également en compte les multiplicités des éléments qui forment chaque structure de simulation. Ces multiplicités placent une contrainte sur le nombre d'éléments autorisés dans chaque structure de simulation.

3.1. Arbre de simulation

Un arbre de simulation (Tree) est composé d'une racine, d'un ou de plusieurs coordinateurs et d'un ou de plusieurs simulateurs. Un coordinateur a des fils qui sont coordinateurs ou simulateurs. La représentation XML correspondante est donnée en Figure 55.

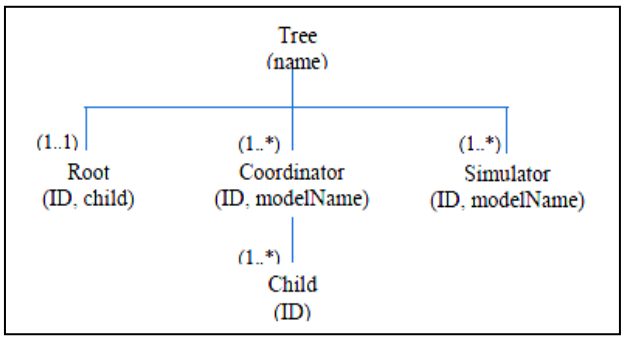


Figure 55. Représentation XML de l'arbre de simulation

3.2. Squelette de simulation

Un squelette de simulation (Skeleton) est une transformation d'arbre de simulation qui introduit des racines supplémentaires et leur affecte des nœuds à contrôler. La Figure 56 donne un exemple de squelette de simulation, et la Figure 57 en donne la représentation XML.

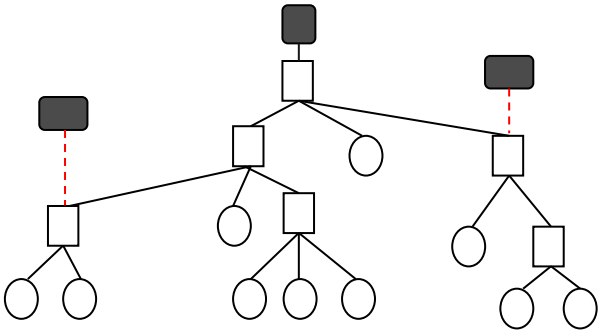


Figure 56. Exemple de squelette de simulation

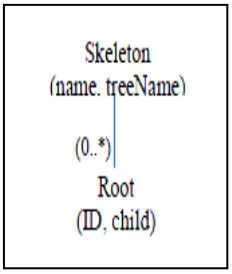


Figure 57. Représentation XML du squelette de simulation

Le principe adopté est que chaque représentation produit au format XML contienne les informations sur la structure courante et fasse référence à la structure de simulation précédente. C'est pourquoi la représentation XML du squelette de simulation contient les informations sur l'arbre initial de simulation. L'avantage de cette approche est de prévenir la perte de l'information sur une quelconque des structures de simulation durant le processus de construction. Il est donc possible de faire des retours en arrière dans le parcours de la méthodlogie SGF (Figure 54) si le choix de la fonction de transformation appliquée à un niveau donné ne s'avère pas satisfaisant.

3.3. Bouquet de simulation

Le bouquet de simulation (Bundle), dont un exemple est donné en Figure 58, est constitué de l'ensemble des processus de simulation. Un processus est un programme d'exécution. Il peut contenir plusieurs nœuds du squelette de simulation, mais il doit contenir au plus un nœud actif. Nous distinguons nœuds actifs et passifs. Les premiers sont des entités concurrentes (à l'image des Threads en java, des Tasks en Ada, etc.). Les seconds sont des entités passives qui ne réagissent que sur sollicitation (à l'image des appels de méthodes d'objet). Le schéma XML du bouquet, donné en Figure 59, fait référence à celui du squelette dont il provient.

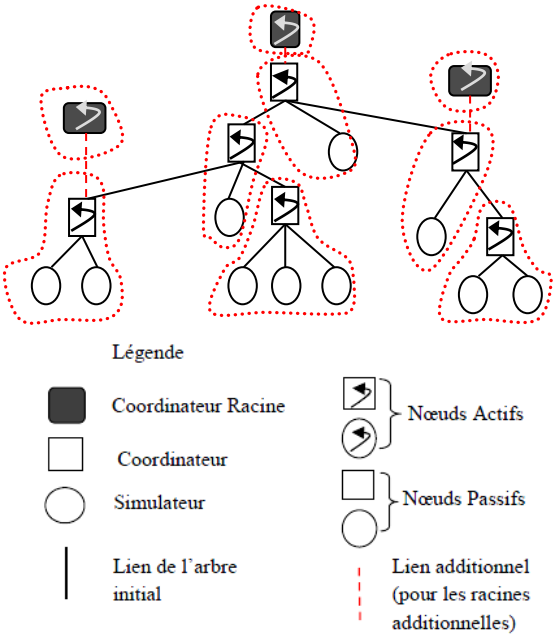


Figure 58. Bouquet de simulation

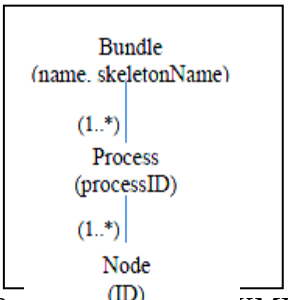


Figure 59. Représentation XML du Bouquet de simulation

3.4. Graphe de simulation

La Figure 60 donne tous les concepts indispensables à la construction d'un graphe de simulation. Pour sa description XML, il est nécessaire d'introduire d'abord la description des processeurs de la plateforme hôte, que nous appelons Machine Virtuelle (MV). Le schéma XML de la MV est donné en Figure 61, et celui du graphe de simulation en Figure 62.

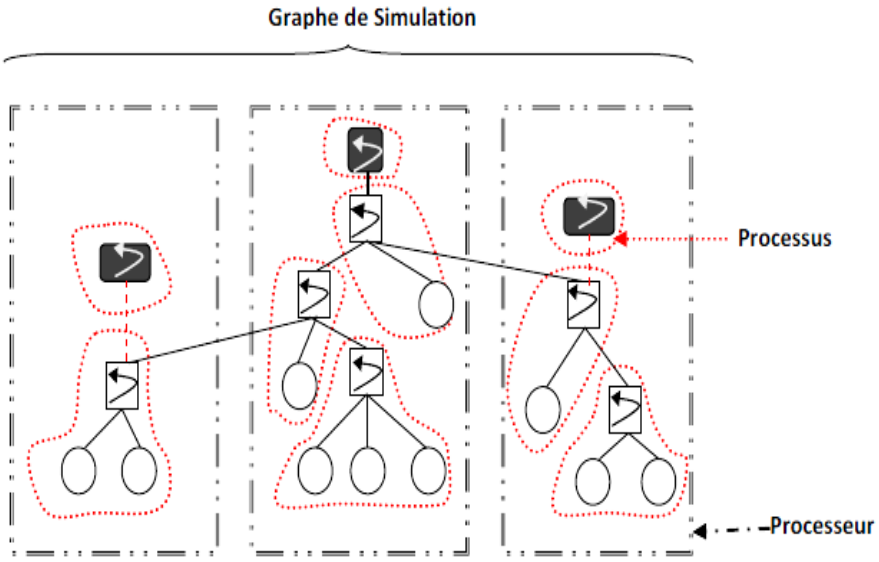


Figure 60. Graphe de simulation dans SGF

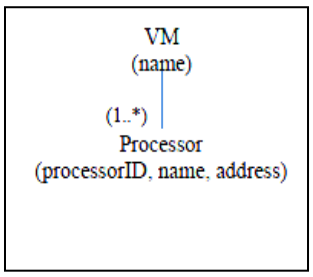


Figure 61. Représentation XML de la VM

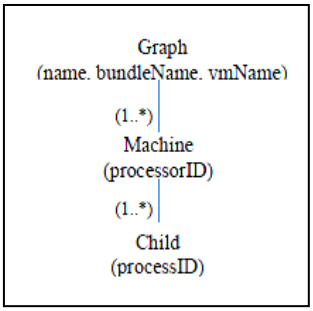


Figure 62. Représentation XML du graphe de simulation

3.5. Méta-modèle du SGF

Le méta-modèle, sous-jacent à l''architecture proposée pour le développement de la SGF, est donnée par la Figure 63. Il permet de vérifier la conformité des PIMs successifs obtenus lors

de la chaîne des transformations vers le graphe de simulation, à une seule et unique référence. Il pourra également servir à tout groupe désireux d'implémenter le MD3SEA de cadre de construction des outils supports permettant d'automatiser la démarche. Le méta-modèle a été spécifié en utilisant EMF/Ecore [Budinsky et al. 2003].

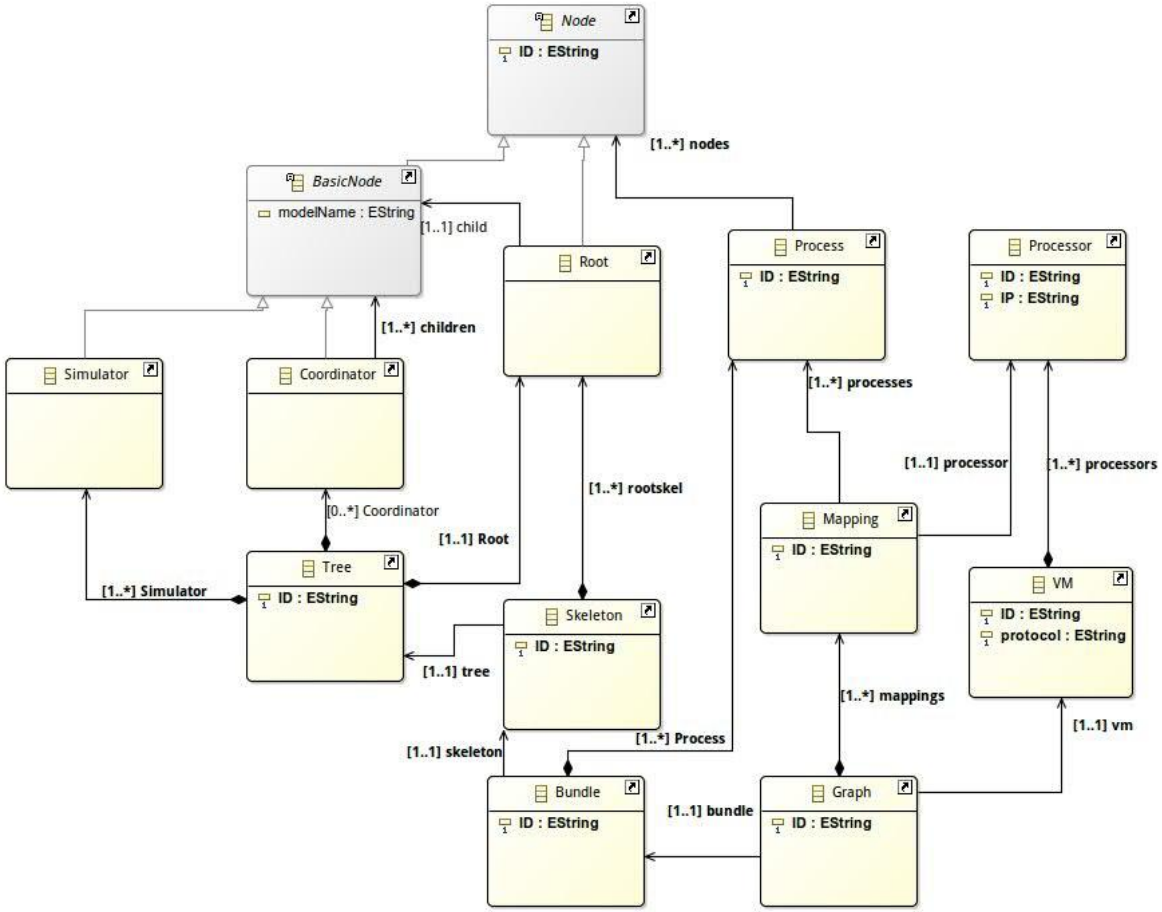


Figure 63. Méta-modèle pour le SGF

4. Framework SimStudio

Dans cette section, nous présentons l'implémentation de la méthodologie proposée dans le framework SimStudio.

4.1. Architecture du Framework

Notre Framework dont l'architecture est basée sur la méthodologie et le méta-modèle exploite les facilités de SimStudio [Traoré 2008]. Basé sur le formalisme DEVS, SimStudio est une plateforme multicouche qui vise à fournir un Framework hautement intégré pour la modélisation et la simulation. Elle présente une architecture logicielle modulaire qui s'appuie sur des plug-ins. Ainsi, chaque opération de la méthodologie proposée est exécutée au moyen d'un plug-in du Framework, comme illustré par la Figure 64. Chaque plugin prend du XML en entrée et produit du XML en sortie. Chaque plug-in apporte une fonctionnalité qui peut être invoquée par l'utilisateur ou réutilisé et étendu par d'autres plug-ins dans le framework.

En outre, les fichiers XML produits représentent les informations sur les structures de simulation et ces informations sont utilisées par les ressources internes. Par exemple, l'éditeur de modèle (Model Editor) est utilisé pour extraire une représentation XML du modèle DEVS à partir d'une description du modèle (capture d'écran de la Figure 65). Le fichier XML produit est utilisé par le générateur d'arbre (Model-to-Tree Generator) pour générer l'arbre DEVS de simulation (Tree). A noter que tout autre éditeur de modèle DEVS qui produit des fichiers XML conformes peut être utilisé comme plugin du framework.

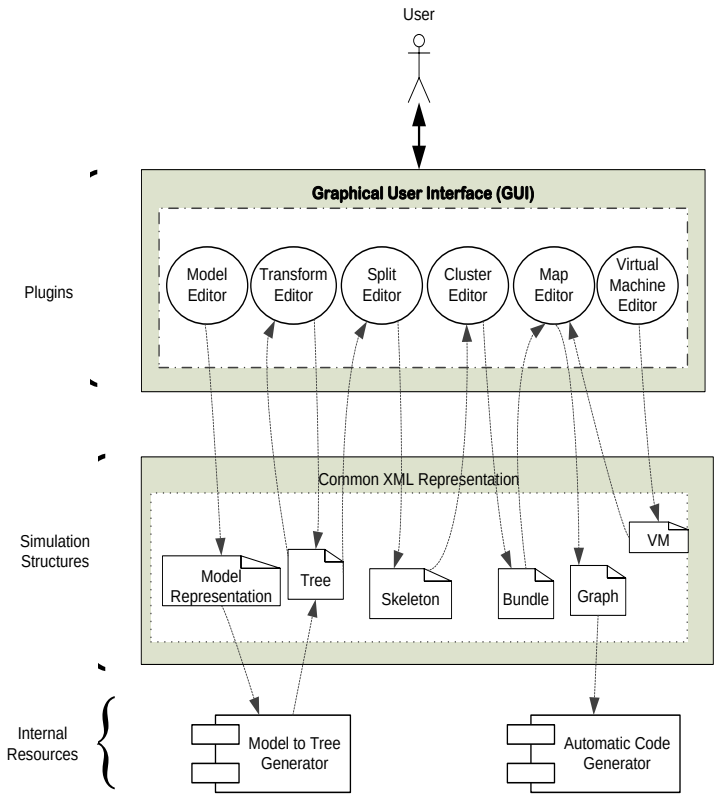


Figure 64. Architecture du Framework SimStudio

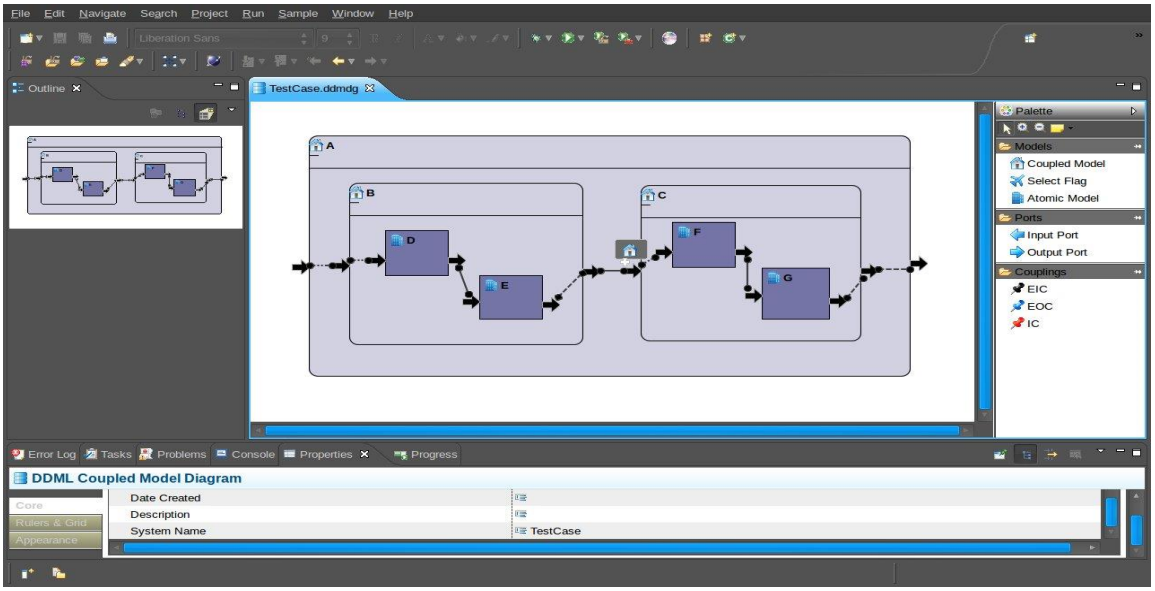
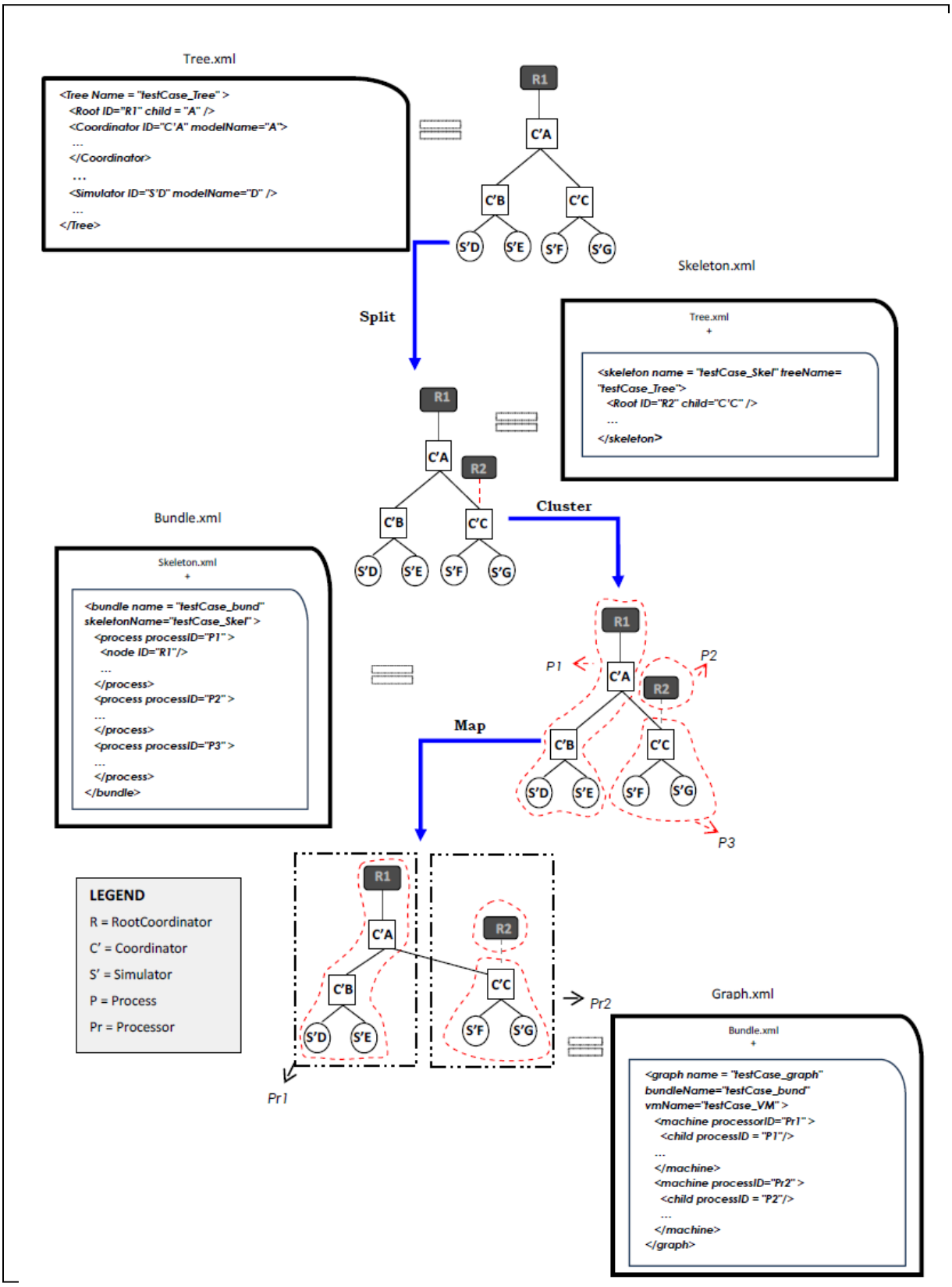


Figure 65. Capture d'écran de l'éditeur de modèle DEVS

Les autres éditeurs permettent à l'utilisateur de définir, de manière visuelle, les fonctions Split, Cluster et Map, entrainant à chaque étape, la production d'un fichier XML. La Figure 66 montre un exemple de parcours, et les fichiers XML générés.



Les représentations XML complètes de toutes les structures de simulation de l'exemple précédent sont présentées par la Figure 67.

Arbre :

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Tree SYSTEM "tree.dtd">
<Tree name="testCase_Tree" xmlns="http://ddml/1.0" >
<Root ID="R1" child="C'A" />
<Coordinator ID="C'A" modelName="A">
<child ID="C'B" />
<child ID="C'C" />
</Coordinator>
<Coordinator ID="C'B" modelName="B">
<child ID="S'D" />
<child ID="S'E" />
</Coordinator>
<Coordinator ID="C'C" modelName="C">
<child ID="S'F" />
<child ID="S'G" />
</Coordinator>
<Simulator ID="S'D" modelName="D" />
<Simulator ID="S'E" modelName="E" />
<Simulator ID="S'F" modelName="F" />
<Simulator ID="S'G" modelName="G" />
</Tree>
```

Bouquet :

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE bundle SYSTEM "bundle.dtd">
<bundle name = "testCase_bund"
skeletonName="testCase_Skel" xmlns =
"/XML_DEVS/skeleton">
        <process processID="P1" >
                <node ID="R1"/>
                <node ID="C'A" />
                <node ID="C'B" />
            <node ID="S'D" />
            <node ID="S'E" />
        </process>
        <process processID="P2" >
                <node ID="R2" />
        </process>
        <process processID="P3" >
                <node ID="C'C" />
                <node ID="S'F" />
                <node ID="S'G" />
        </process>
</bundle>
```

Squelette :

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE skeleton SYSTEM "skeleton.dtd">
<skeleton name = "testCase_Skel" treeName=
"testCase_Tree" xmlns = "/XML_DEVS/transform">
        <Root ID="R2" child="C'C" />
</skeleton>
```

Machine Virtuelle :

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE VM SYSTEM "vm.dtd">
<VM name="testCase_VM">
  <Processor processorID="P_1" name = "Virtual_1"
address="192.168.110.1"/>
  <Processor processorID="P_2" name = "Virtual_2"
address="192.168.110.2"/>
</VM>
```

Graphe :

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE graph SYSTEM "graph.dtd">
<graph name = "testCase_graph"
bundleName="testCase_bund"
vmName="testCase_VM" xmlns:bundle =
"/XML_DEVS/bundle" xmlns:vm =
"/XML_DEVS/vm">
        <vm:machine processorID="Pr1" >
                <bundle:child processID = "P1"/>
        </vm:machine>
        <vm:machine processorID="Pr2" >
                <bundle:child processID = "P2"/>
                <bundle:child processID = "P3"/>
        </vm:machine>
</graph>
```

Figure 67. Exemple de structures XML pour graphe de simulation

4.2. Projection sur calques

Afin de faciliter d'automatiser le travail de transformation de simulation séquentielle en simulation répartie, nous avons introduit dans SimStudio la notion de projection sur calque. L'idée est de faire remonter à la modélisation, les choix de projection de l'arbre de simulation sur un graphe de simulation. Les calques sont des abstractions des manteaux de la même

manière que les modèles atomiques et couplés sont des abstractions respectivement des simulateurs et coordinateurs.

La Figure 68 explique le principe de modélisation par calques. La fonction fSGen est celle qui définit la sémantique opérationnelle de DEVS (nous la connaissons donc). La fonction fSGF est celle que devrait construire l'utilisateur pour paralléliser son simulateur séquentiel (dans le cas de l'approche par manteaux, nous la connaissons également, même si une stricte formalisation mathématique n'a pas été présentée dans ce mémoire). L'idée de la projection par calque est de faire construire par le modélisateur la fonction fCalq, qui nous permet alors d'obtenir le graphe de simulation, car la fonction fPGen est définie par la relation :

$$fSGF \; o \; fSGen = fPGen \; o \; fCalq$$

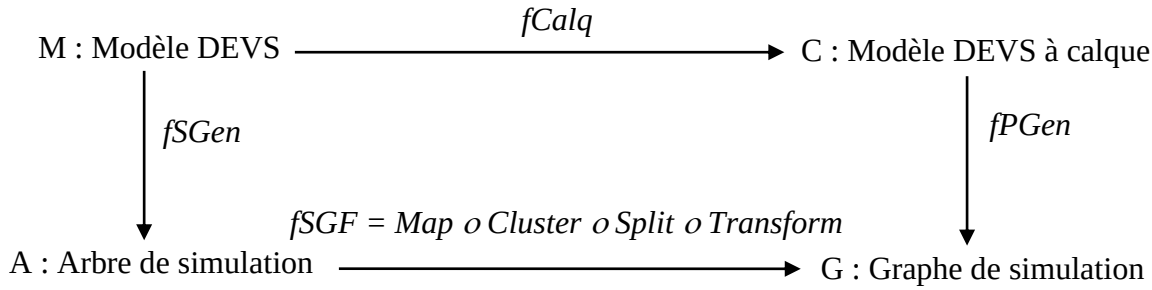


Figure 68. Principe de la projection sur calques

Le principe de projection par calque est le suivant : le modèle couplé global est projeté, par l'utilisateur, sur plusieurs calques en fonction du nombre de ressources de calcul disponibles (le nombre de calques étant égal au nombre de ressources de calcul). Cette projection sur calques est spécifiée de la façon suivante :

$U =< M, E, \{M_k\}_{k \in E}, Rem >$ *avec*

- $M =< X, Y, D, \{M_d\}_{d \in D}, EIC, EOC, IC >$ *est un modèle couplé*
- M_k *est une projection de M*
- *E est l'ensemble des noms des calques*
- $Rem = \{(M_i, Pr_j)/i \in E\}$
- Pr_j *est une ressource de calcul*

La Figure 69 montre une projection sur calques du modèle M0 (le modèle sans calque est à gauche et le modèle calqué est à droite). Deux calques sont utilisés : le premier calque couvre M0 sans le sous composant M2 et le deuxième calque recouvre le sous composant M2.

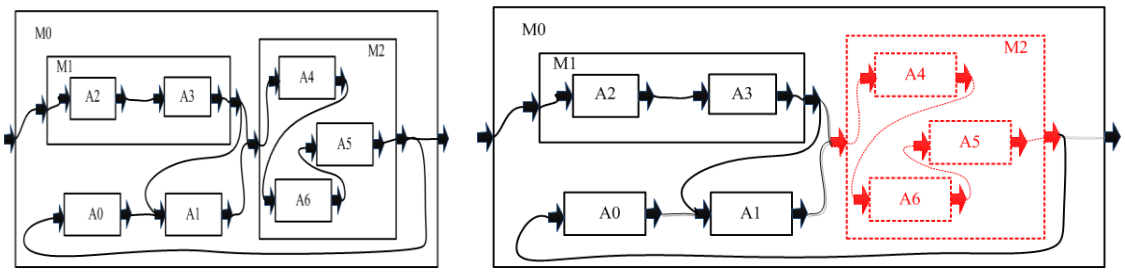


Figure 69. Modèle sans calque et avec calques

La spécification de cet exemple est la suivante :

$U =< M, E, \{M_k\}_{k \in E}, Rem >$ avec

- $M = \{M_0\}$,

- $E = \{a,\ b\}$,
- $M_a = \{M_1, A_0, A_1\}$,
- $M_b = \{M_2\}$
- $Rem = \{(M_a,\ Pr_1); (M_b,\ Pr_2)\}$,

Une fois les calques définis, chacun d'eux peut être décrit séparément. Les composants distants/absents dans un calque sont remplacés par des * (avec des références aux noms des calques correspondants), comme le montre la Figure 70. La superposition des calques doit évidemment redonner le modèle global de départ.

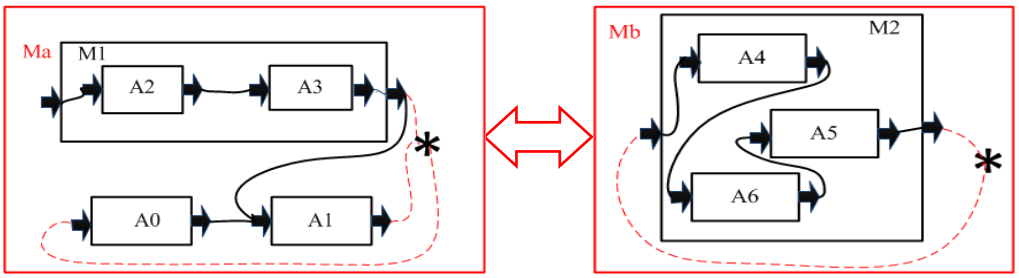


Figure 70. Calques Ma et Mb

La Figure 71 montre un exemple de génération de graphe de simulation avec manteaux, à partir d'une spécification de modèle calqué.

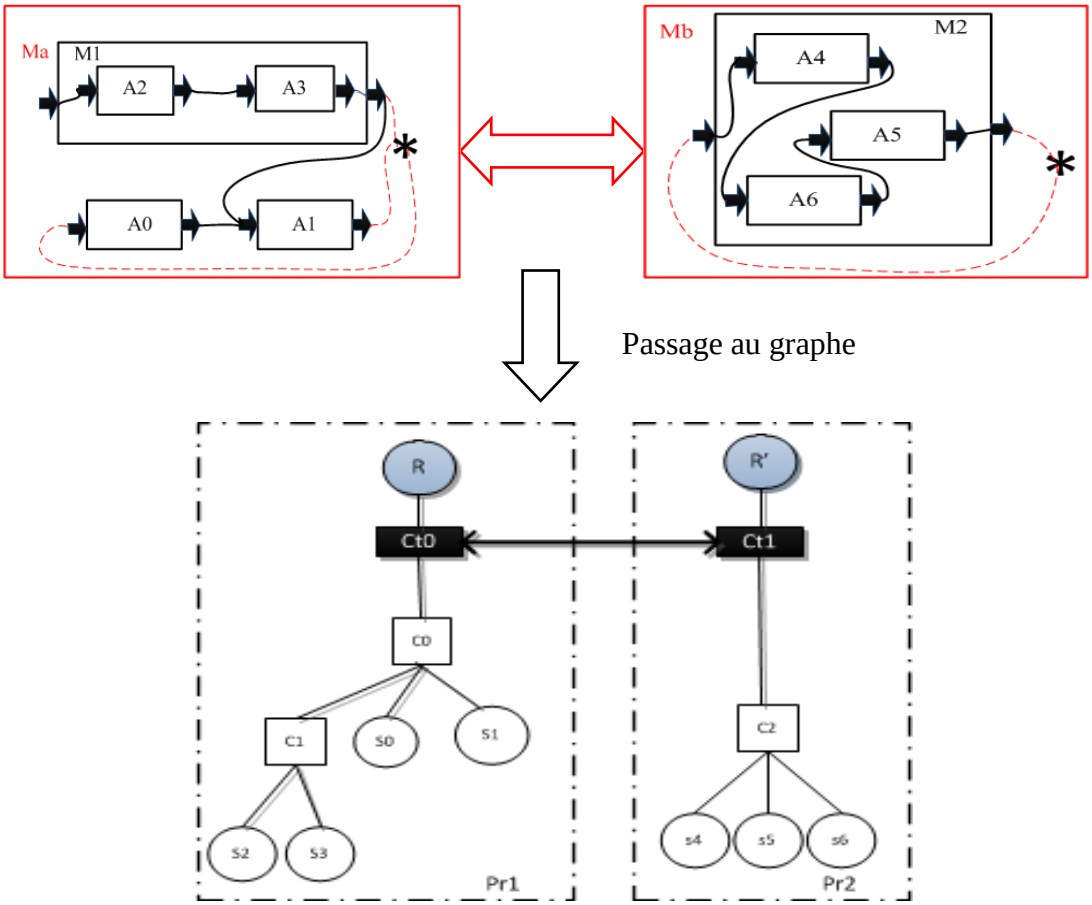


Figure 71. Exemple de passage des calques au graphe de simulation

5. Conclusion

Dans ce chapitre, nous avons proposé un cadre d'Ingénierie Dirigée par les Modèles dédié à la parallélisation de simulateurs DEVS. Elle est enracinée dans les principes du MDA et propose d'utiliser DEVS, XML et UML comme les DSLs de l'architecture proposée. La modélisation DEVS produit la connaissance métier (CIM), et une série de transformations, structurées dans une méthodologie appelée SGF, permet de générer progressivement le graphe de simulation correspondant (PIM), qui à son tour sera transformé en code exécutable lorsque les mécanismes de communication inter processeurs sont définis (PSM).

Après avoir présenté les concepts et principes de notre architecture, appelée MD3SEA, nous en avons proposé une implémentation dans notre framework de M&S, SimStudio, qui adopte la démarche préconisée pour mettre en place l'approche par manteau que nous avons définie dans le chapitre précédent. Enfin, nous avons introduit une approche de spécification par calque, faisant remonter le travail de répartition de simulateur DEVS au niveau conceptuel.

Chapitre V.
CONCLUSION GENERALE

Dans cette thèse, nous proposons une démarche permettant de paralléliser des implémentations DEVS existantes, en évitant de modifier le code des composants de bases (simulateur, coordinateur et coordinateur racine), mais en introduisant des composants manteaux. Cette approche nous permet d'éviter que de nouveaux types de messages apparaissent dans les composants existants. Les messages classiques (qui circulent entre les simulateurs et les coordinateurs) de la simulation séquentielle DEVS restent inchangés. Les nouveaux types de message, introduits pour la simulation répartie avec DEVS ne circulent qu'entre les manteaux. En ce sens, notre approche se distingue de ce que la littérature propose. Les études de performance que nous avons effectuées, montrent que l'approche est viable en termes de vitesse d'exécution. Toutefois, un plus large éventail de tests est nécessaire. En effet, plusieurs critères méritent d'être variés pour faire une analyse de sensibilité probante : profondeur et autres caractéristiques fines des modèles simulés, nombre de processeurs en jeu, architecture hôte (grille, cluster, nuage…), mécanismes de communication inter processeurs (Web services, multiprocesseurs…, partition, etc.

Nous voyons dans notre travail trois contributions majeures :

- La définition d'algorithmes de parallélisation de simulateur DEVS, structurés en composants modulaires injectables (Manteaux), en autant d'exemplaires que nécessaire, dans une implémentation séquentielle existante.
- La définition d'une démarche générique de systématisation de la mise en œuvre de cette parallélisation (MD3SEA), applicable à n'importe quelle implémentation séquentielle DEVS.
- Le développement d'un cadriciel de simulation répartie DEVS (SimStudio), basé sur l'injection de ces composants nouveaux dans le code d'un environnement initialement conçu pour une exécution séquentielle.

Ce travail est loin d'être achevé. Nous avons montré son opérationnalité, il nous reste à travailler sur son automatisation complète. En particulier, le passage au PSM demande un important effort de formalisation. Par ailleurs, les technologies que nous avons utilisées pour faire notre banc d'essai (CORBA, MPI) sont aujourd'hui considérées comme obsolètes. De nouvelles voies s'ouvrent dans ce domaine vers les Web services, les protocoles mobiles et le Calcul Haute Performance. Nous nous devons de faire évoluer les protocoles de communication inter processeurs de notre framework vers ces nouvelles solutions.

Une autre limite de notre approche par manteaux est l'absence d'optimisation. Si l'objectif est de gagner de plus en plus en temps d'exécution en prenant appui sur les architectures nouvelles, le protocole de communication adopté n'est de toute évidence pas optimisé en termes de messages de simulation échangés entre les nœuds de calcul. Un compromis entre l'allègement de la charge de messages échangés et la volonté de ne pas altérer les algorithmes initiaux de simulation DEVS doit être trouvé. Une autre piste est de se servir de l'Ingénierie Dirigée par les Modèles pour obtenir automatiquement cette optimisation (avec probablement des règles d'altération des codes existants, mais systématisées).

Nos travaux futurs incluent :

- Un large banc d'essai pour étudier les performances de l'approche par manteaux, et ce non seulement en fonction des critères matériels et techniques (architecture hôte, protocole de communication inter processeurs, stratégie d'implémentation des composants, partitions…), mais aussi en fonction des critères de modélisation (caractéristiques des modèles DEVS).

- Des efforts d'optimisation de l'approche par manteaux, et ce dans les cas pessimistes et optimistes.
- La finalisation d'un outillage logiciel complet, permettant une totale automatisation des processus proposés, de la modélisation au déploiement réparti et à l'expérimentation.

BIBLIOGRAPHIE

[Adegoke et al. 2011] Adegoke A., Amadou I., Togo H., Traoré M. K. 2011. "A Methodology for the DEVS Simulation Graph Construction." *Proceedings of the 23rd EMSS, September 12-14, 2011,* Rome, Italy.

[Adegoke et al. 2013] Adegoke A., Togo H., Traoré M.K. 2013. "A Unifying Framework for Specifying DEVS Parallel and Distributed Simulation Architectures" Special Issue on Advancing Simulation Theory and Practice with Distributed Computing, SIMULATION 89, no. 11: 1293-1309.

[Atkinson & Kuhne 2003] Atkinson C. and Kuhne T. 2003. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5) :36–41.

[Bézivin 2004] Bézivin J. 2004. In Search of a Basic Principle for Model Driven Engineering. *The European Journal for the Informatics Professional*, 2:21–24.

[Blanc 2005] Blanc X. 2005. MDA en action. *Eyrolles*, March.

[Bryant 1977] Bryant R. E. 1977. Simulation of Packet Communication Architecture Computer Systems. *Massachusetts Institute of Technology*. Cambridge, MA, USA.

[Budinsky et al. 2003] Budinsky F., Steinberg D., Ellersick R. 2003. Eclipse Modeling Framework: A Developer's Guide. *Addison-Wesley Professional*.

[Burmester et al. 2004] Burmester S., Giese H., Niere J., Tichy M., Wadsack J.P., Wagner R., Wendehals L., Zündorf A. 2004. Tool Integration at the Meta-Model Level within the FUJABA Tool Suite**.** *International Journal on Software Tools for Technology Transfer (STTT)*, 6(3) : 203-218, Springer Verlag.

[Chandy & Misra 1979] Chandy K.M., Misra J. 1979. Distributed Simulation: A case Study in Design and Verification of Distributed Programs. *IEEE transactions on Software Engineering*, 5(5) :440-452.

[Chandy & Misra 1981] Chandy K.M., Misra J. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *ACM*, 24(4) :198-206.

[Chandy et al. 1983] Chandy K.M., Misra J., Haas L. 1983. Distributed Deadlock Detection. *ACM Trans. Comput. Syst.* I,2, 144-156.

[Cheon et al. 2004] Cheon S., Seo C., Park S., Zeigler B. 2004. Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Network System. *Advanced Simulation Technologies Conference – Design, Analysis, and Simulation of Distributed Systems Symposium*. Arlington, USA.

[Chow 1996] Chow A. 1996. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator. *SCS Transaction on Simulation* 13(2): 55-102.

[Christensen & Zeigler 1990] Christensen E.R. and Zeigler B. P., 1990. Distributed Discrete Event Simulation: Combining DEVS and Time Warp. *Proceedings of the SCS Eastern Multiconference on AI and Simulation Theory and Applications.*

[Combemale 2008] Combemale B. 2008. Approche de Métamodélisation pour la Simulation et la Vérification de Modèle – Application à l'Ingénierie des Procédés. *PhD thesis, INPT ENSEEIHT*, July.

[Csarnecki & Helsen 2003] Czarnecki K. and Helsen S. 2003. Classification of Model Transformation Approaches. *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

[Davila & Uzcagegui 2000] Davila J. and Uzcategui M. 2000. GALATEA: A Multi-agent Simulation Platform. *International Conference on Modeling, Simulation and Neural Networks*. MSSN'2000.

[Delara & Vangheluwe 2002] De Lara J., Vangheluwe H. 2002. ATOM3: A tool for Multi-formalism Modeling and Meta-modeling. *European Joint Conference on Theory and Practice of Software*. Grenoble, France.

[Dijkstra & Scholten 1980] Dijkstra E.W. and Scholten, C.S. 1980. Termination Detection for Diffusing Computations. *Proc. Lett.* 12, pp. 1-4.

[Ehrig et al. 2005] Ehrig K., Ermel C., Hänsgen S., Taentzer G. 2005. Towards Graph Transformation Based Generation of Visual Editors Using Eclipse. *Electronic Notes in Theoretical Computer Science*, 127(4).

[Farail et al. 2006] Farail P., Gaufillet P., Canals A., Le Camus C., Sciamma D., Michel P., Crégut X., Pantel M. 2006. The TOPCASED Project: a Toolkit in OPen Source for Critical Aeronautic Systems Design. *3rd European Congress on Embedded Real Time Software (ERTS)*, Toulouse, France, January.

[Filippi et al. 2002] Filippi J.B., Bernardi F., Delhom M. 2002. The JDEVS Modeling and Simulation Environment. *Integrated Assessment and Decision Support Conference (IEMSS'02)*. Lugano, Switzerland, pp. 283-288.

[Fujimoto 1990] Fujimoto R.M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10): 30-53.

[Fujimoto 2000] Fujimoto R.M. 2000. Parallel and Distributed Simulation Systems. *A Wiley-Interscience publication*.

[Garredu et al. 2014] Garredu S., Vittori E., Santucci J-F., Poggi B. 2014. A Survey of Model-Driven Approaches Applied to DEVS. A Comparative Study of Metamodels and Transformations. *International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*. Vienna, Austria, ISBN: 978-989-758-060-4, pp. 179-187.

[Genaud & Rattanapoka 2007] Genaud S. and Rattanapoka C. 2007. P2P-MPI: A Peer-to-Peer Framework for Robust Execution of Message Passing Parallel Programs on Grids. *Journal of Grid Computing*, 5(1):27-42, Springer, ISSN:1570-7873 2007.

[Gipps 1981] Gipps P.G. 1981. A Behavioral Car-following Model for Computer Simulation. *Transportation Research Part B*, Vol. 15, pp 105-111.

[Gray et al. 2007] Gray J., Fisher K., Consel C., Karsai G., Mernik M., Tolvanen J.P. 2007. DSLs: the good, the bad, and the ugly. *23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Nashville, TN, USA.

[Himmelspach & Uhrmacher 2006] Himmelspach J. and Uhrmacher A.M. 2006. Sequential Processing of PDEVS Models. *EMSS*, Barcelona, Spain, pp.239–244.

[Jafer & Wainer 2010] Jafer S., and Wainer G.L. 2010. Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study. *Summer Simulation Conference (SummerSim10), SCSC symposium*, pp. 342-350.

[Janousek et al. 2006] Janousek V., Polásek P., Slavícek P. 2006. Towards DEVS Meta Language. *ISC*. Zwijnaarde, BE, pp. 69-73.

[Jefferson 1985] Jefferson D.R. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7(3): 404-425.

[Jouault & Kurtey 2005] Jouault F. and Kurtev I. 2005. Transforming Models with ATL. Lecture Notes in Computer Science, 3884 : 128–138, Springer.

[Kim & Kang 2004] Kim K. and Kang W. 2004. CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One. *International Conference on Computational Science and Its Applications (ICCSA)*. Assisi, Italy.

[Kim et al. 1998] Kim K., Kim T., Park K. 1998. Hierarchical Partitioning Algorithm for Optimistic Distributed Simulation of DEVS Models. *Journal of Systems Architecture*, 44, pp. 433–455.

[Kleppe et al. 2003] Kleppe A., Warmer J., Bast W. 2003. MDA Explained. The Model Driven Architecture : Practice and Promise. *Addison-Wesley*.

[Kofman et al. 2003] Kofman E., Lapadula M., Pagliero E. 2003. PowerDEVS: a DEVS-based Environment for Hybrid System Modeling and Simulation. *Technical Report LSD0306*, University Nacional de Rosario.

[Koné 2015] Koné Y. and Traoré M.K. 2015. Study and Spatio-temporal Predictions of Urban Transport Systems. *Rapport Interne GREP/RR-03-15*, Vichy, France.

[Leroudier 1980] Leroudier J. 1980. La Simulation à Evénements Discrets. *Editions Hommes et Techniques*.

[Liu & Wainer 2007] Liu Q. and Wainer G. 2007. Parallel environment for DEVS and Cell-DEVS models. *SIMULATION* 83(6) : 449-471.

[Miller & Thorpe 1983] Miller D.C. and Thorpe J.A. 1983. SIMNET: The Advent of Simulator Networking. *IEEE* 83(8):1114-1123.

[Misra 1986] Misra J. 1986. Distributed Discrete Event Simulation. *Computing Surveys* 18(1):39-65. 1986

[Muller et al. 2005] Muller P-A., Fleurey F., Jézéquel J-M. 2005. Weaving Executability into Object-Oriented Meta-Languages. *MODELS/UML'2005, LNCS*, Montego Bay, Jamaica, October. Springer.

[Nutaro 2010] Nutaro J. 2010. Building Software for Simulation: Theory, Algorithms, and Applications in C++. *Wiley*. ISBN 0-470-41469-3.

[OMG 2000] Object Management Group. 2000. The Common Object Request Broker: Architecture and Specification. *Object Management Group, 2.4 edition*.

[OMG 2006] Object Management Group. 2006. Meta-object facility 2.0 core specification. Available at http://www.omg.org/spec/MOF/2.0/

[Park 2003] Park S. 2003. Cost-based Partitioning for Distributed Simulation of Hierarchical, Modular DEVS Models. *PhD thesis*, Department of Electrical and Computer Engineering, University of Arizona.

[Praehofer et al. 1999] Praehofer H., Sametinger J., Stritzinger A. "Discrete Event Simulation Using the JavaBeans Component Model. *International Conference on Web-Based Modeling & Simulation*. San Francisco, CA. USA.

[Sarjoughian & Zeigler 1998] Sarjoughian H.S. and Zeigler B. 1998. DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment. *International Conference on Web-Based Modeling and Simulation*. Vol. 5, pp. 29-36. San Diego, CA. USA. 1998.

[Seo et al 2004] Seo C., Park S., Kim B., Cheon S., Zeigler B. 2004. Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment". *Advanced Simulation Technologies Conference (ASTC)*. Arlington, VA. USA.

[Seong et al. 1995] Seong Y.R., Jung S.H., Kim T.G., Park K.H. 1995. Parallel Simulation of Hierarchical Modular DEVS Models: A Modified Time Warp Approach. *International Journal of Computer Simulation 5(3).*

[Tolvanen & Rossi 2003] Tolvanen J-P., Rossi M. 2003. MetaEdit+: Defining and Using Domain-Specific Modeling Languages and Code Generators. *18th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*, Anaheim, California, October 26-30, pp. 92-93.

[Traore 2008] Traoré, M.K. 2008. SimStudio. A Next Generation Modeling and Simulation Framework. SIMUTools'08, ISBN 978-963-9799-20-21, Marseille, France, March 3-7.

[Uhrmacher 2001] Uhrmacher A.M., 2001. Dynamic Structures in Modeling and Simulation: a Reactive Approach. TOMACS, 11(2): 206-232.

[W3C 2015] W3C. 2015. XSL Transformations (XSLT) *http://www.w3.org/TR/xslt*, October.

[Wainer 2002] Wainer G. 2002. CD++: a Toolkit to Develop DEVS Models. *Software – Practice and Experience*. Vol. 32, pp. 1261-1306.

[Wainer 2009] Wainer G. 2009. Discrete-event Modeling and Simulation: a Practitioner's Approach. *CRC Press*, New York.

[Wainer et al. 2008] Wainer G.A., Madhoun R., Al-Zoubi K. 2008. Distributed Simulation of DEVS and Cell-DEVS Models in CD++ Using Web-Services. Simulation Modelling Practice and Theory. 16(9) : 1266–1292.

[Zeigler & Kim 1993] Zeigler B. and Kim J. 1993. Extending the DEVS-Scheme Knowledge-based Simulation Environment for Real-time Event-based Control. *IEEE Transactions on Robotics and Automation*. 9(3) : 351-356.

[Zeigler 1976] Zeigler B.P. 1976. Theory of Modeling and Simulation. *Wiley-Interscience*, New York.

[Zeigler et al. 1996] Zeigler B., Moon Y., Kim D., Kim J.G. 1996. DEVS-C++: A High Performance Modeling and Simulation Environment". *The 29th Hawaii International Conference on System Sciences*.

[Zeigler et al. 1999 a] Zeigler B., Ball G., Cho H., Lee J., Sarjoughian H. 1999. The DEVS/HLA Distributed Simulation Environment and its Support for Predictive Filtering. *SIMULATION: Special Issue on The High Level Architecture*. 73(4).

[Zeigler et al. 1999 b] Zeigler B., Kim D., Buckley S. 1999. Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment. *Winter Simulation Conference*.

[Zeigler et al. 2000] Zeigler B.P., Kim T.G., Praehofer H. 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. *Academic Press, 2nd Edition*, New York.

[Zhang et al. 2006] Zhang M., Zeigler B., Hammonds P. 2006. DEVS/RMI – An Auto-adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies. *DEVS Integrative M&S Symposium (DEVS'06)*. Huntsville, Alabama, USA.